TRABALHO DE GRADUAÇÃO

PROPOSTA DE PLATAFORMA INERCIAL E SIMULADOR 3D PARA PERICIAR ACIDENTES DE TRÂNSITO

Hudson Pereira Ramos

Vanessa Oliveira Lucena

Brasília, julho de 2017

ENGENHARIA
MECATRÔNICA
UNIVERSIDADE DE BRASÍLIA

UNIVERSIDADE DE BRASILIA
Faculdade de Tecnologia
Curso de Graduação em Engenharia de Controle e Automação

TRABALHO DE GRADUAÇÃO

PROPOSTA DE PLATAFORMA INERCIAL E SIMULADOR 3D PARA PERICIAR ACIDENTES DE TRÂNSITO

Hudson Pereira Ramos

Vanessa Oliveira Lucena

Relatório submetido como requisito parcial de obtenção
de grau de Engenheiro de Controle e Automação

Banca Examinadora

Prof. Ricardo Zelenovsky, ENE/UnB _____
Orientador

Prof. Daniel Café, ENE/UnB _____
Examinador interno

Eng. Vinícius Lima _____
Examinador externo

Brasília, julho de 2017

FICHA CATALOGRÁFICA

PEREIRA RAMOS, HUDSON;LUCENA, VANESSA DE OLIVEIRA
Proposta de Plataforma Inercial e Simulador 3D para periciar acidentes de trânsito

[Distrito Federal] 2017.

vi, 76p., 297 mm (FT/UnB, Engenheiro, Controle e Automação, 2017). Trabalho de Graduação – Universidade de Brasília. Faculdade de Tecnologia.

1. Plataforma inercial para veículos
2. Perícia para acidentes de trânsito
3. Simulador de acidentes de trânsito

I. Mecatrônica/FT/UnB

REFERÊNCIA BIBLIOGRÁFICA

PEREIRA RAMOS, HUDSON; LUCENA, VANESSA DE OLIVEIRA, (2017). PROPOSTA DE PLATAFORMA INERCIAL E SIMULADOR 3D PARA PERICIAR ACIDENTES DE TRÂNSITO. Trabalho de Graduação em Engenharia de Controle e Automação, Publicação FT.TG-$n°$014/2017, Faculdade de Tecnologia, Universidade de Brasília, Brasília, DF, 76p.

CESSÃO DE DIREITOS

AUTORES: Hudson Pereira Ramos e Vanessa Oliveira Lucena

TÍTULO DO TRABALHO DE GRADUAÇÃO:Proposta de Plataforma Inercial e Simulador 3D para periciar acidentes de trânsito

GRAU: Engenheiro de Controle e Automação ANO: 2017

Pereira Ramos, Hudson; Lucena, Vanessa de Oliveira

71000-000 Brasília – DF – Brasil.

Dedicatórias

Ao meu pai Edemilson Alves Pereira e à minha mãe, Alda Messias Ramos.

Hudson Pereira Ramos

À minha mãe Rita de Cássia Oliveira Nascimento e ao meu pai, Edvaldo Dias Lucena..

Vanessa Oliveira Lucena

Agradecimentos

Agradeço a todos aqueles que ajudaram na realização deste Trabalho de Graduação, me proporcionando forças para que eu não desistisse de ir atrás do que eu buscava para minha vida. Diversos obstáculos foram me impostos durante esses anos de graduação, principalmente quando tive que mudar de curso. Porém, tive uma base familiar sólida, em especial meus pais, Edemilson Alves Pereira e Alda Messias Ramos, que me fornecerem as condições para que eu me dedicasse todos esses anos à Engenharia.
Sou muito grato também ao Pedro Sampaio, colega de curso que me indicou como orientador o Ricardo Zelenovsky, uma pessoa ímpar, cheia de dedicação e comprometimento.
Também quero agradecer à minha colega de curso, Vanessa Lucena, com quem escrevi este trabalho, por ser uma pessoa compreensiva e com determinação.
Por fim e não menos importante, gostaria de agradecer ao Vinícius Lima, que deu início à proposta da plataforma inercial e sempre esteve disposto a ajudar em nosso trabalho.

Hudson Pereira Ramos

Gostaria de agradecer à minha família, aos meus amigos e todos aqueles que me ajudaram durante esses anos de faculdade. Cito principalmente minha mãe, que me apoiou em todos os momentos da minha vida e foi a grande responsável por ter chegado até aqui.
Agradeço o orientador Ricard Zelenovsky que sempre se mostrou presente e disposto a ajudar e à minha dupla, Hudson Pereira, que se mostrou uma pessoa muito dedicada.

Vanessa Oliveira Lucena

RESUMO

A motivação para o desenvolvimento deste projeto reside em fornecer informações que facilitem a reconstrução de acidentes de trânsito que envolvam veículos automotores, visando ajudar na identificação da sua real causa e de eventos envolvidos durante a colisão. Por isso, este trabalho apresenta uma metodologia completa para a construção de uma plataforma inercial para auxílio em perícias que colete dados continuamente sobre aceleração, giro e posição geográfica, e quando identificado a ocorrência de uma batida, tais dados sejam armazenados em uma memória não volátil. Além disso, é abordado o desenvolvimento de um *software* que realiza a leitura das informações armazenadas e simula os movimentos do carro durante a batida.

Palavras Chave: Plataforma inercial e perícia de acidentes de trânsito.

ABSTRACT

The motivation for this study is to provide information to help a vehicular accident reconstruction and aim to identify the causes and events occurred during the vehicle collision. This paper therefore presents a complete methodology to build an Inertial Movement Unity (IMU), to be used in forensic vehicles collisions, which collects acceleration, gyro and position data in real time and when a car crash is detected, the data is stored in a non-volatile memory. Furthermore, it shows the development of a *software*, which uses the data from the IMU to make a three dimensional collision reconstruction.

Keywords: Inertial Movement Unity (IMU), vehicular accident reconstruction

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

Capítulo 1

Introdução

1.1 Contextualização

Acidentes de trânsito causam por ano mais 1,2 milhões de mortes mundialmente, o que acarreta um grande impacto social e econômico em seus agentes envolvidos, de forma direta ou indireta. Cada país tem um gasto de aproximadamente 3% de seu PIB com acidentes de trânsito [3]. No Brasil, apesar dos avanços no Código de Trânsito Brasileiro (CTB) vigente, que prevê leis mais duras e, mesmo com os avanços na engenharia de tráfego, de acordo com a OMS, o Brasil é o quarto país com mais mortes no trânsito na América [4].

Por muito tempo, uma análise estatística dos dados obtidos em acidentes de trânsito tem sido feita para auxiliar no desenvolvimento de carros mais seguros e na criação de políticas públicas que minimizem a quantidade de acidentes de trânsito [5]. No Brasil, acidentes de trânsito com lesão corporal exigem por lei, a necessidade de uma perícia criminal, que busque identificar de forma técnica o modo pelo qual ocorreu o acidente e os fatores que foram decisivos ou contribuíram para tal acidente.

De uma forma mais técnica, de acordo com [6], define-se como acidentes de trânsito eventos do tipo colisões, atropelamentos, capotamentos, tombamentos, abalroamentos, quedas e choques contra obstáculos fixos. Desse modo, analisar as causas de acidentes de trânsito se torna uma tarefa complexa, que envolve muitas vertentes da física, principalmente da mecânica, que engloba a cinemática, dinâmica, estática, etc.

Existem vários vestígios que permitem ao perito criminal investigar as causas do acidente de trânsito, como fricção, fragmentos, frenagem e derrapagem [7]. É com a presença dos últimos dois vestígios, que normalmente a perícia usa as marcas pneumáticas geradas pelos veículos para identificar a velocidade dos mesmos no momento do acidente [8]. Porém, veículos com freios ABS (*Antiblockier-Bremssystem*), que é um sistema que evita o travamento de rodas, tem a tendência de não apresentarem marcas de frenagem. A resolução N° 308/11, alterada pelas resoluções N° 596/16 e N° 395/11 do Conselho Nacional de Trânsito (CONTRAN), obriga o uso de freios ABS para modelos fabricados no Brasil a partir de 2014 [9], o que dificulta ainda mais o uso de marcas de frenagem para reconstrução de acidentes de trânsito.

Outra alternativa para investigar as causas do acidente de trânsito seria utilizar dispositivos EDR's (*Event Data Recorders*), como os módulos de *airbags*, que muitas vezes armazenam diversas informações, entre elas, a velocidade do veículo antes e depois de uma colisão. Porém, como mostrado em [8], no Brasil, não existe uma padronização para a forma de armazenamento de dados nesses módulos e muitas vezes, somente os próprios fabricantes conseguem de fato acessar esses dados. É nesse contexto que se torna importante a criação de uma plataforma inercial que possa ser usada em diferentes tipos de veículos, barata, de fácil instalação e com uma forma simples de acesso de dados.

1.1.1 Simuladores 3D

De acordo com [10], ao longo das últimas décadas o desenvolvimento de recurso da computação gerou um avanço significativo em diversas áreas do conhecimento. Com isso, foi possível produzir computadores com maior poder de processamento e, então, obter recursos gráficos e ambientes virtuais com aplicações práticas em várias áreas.

Atualmente simuladores 3D são utilizados para inúmeras finalidades, desde a simulação de brinquedos em parques de diversões até para estudos médicos em centros de pesquisa.

1.2 Definição do Problema

Em qualquer acidente de trânsito que resulte em morte, pessoas gravemente feridas ou mesmo com danos ao patrimînio, a pericia é acionada. Quando isso ocorre, o Instituto de Criminalística envia ao local um perito criminal e um fotógrafo, que são responsáveis por colher evidências, verificar as marcas de pneus no chão, a posição dos carros e determinar a velocidade provável dos veículos. Depois disso, todo o material coletado e informações adquiridas são levados para algum laboratório de física da polícia técnico-científica, onde serão analisados. Dependendo do caso, a expedição do laudo conclusivo pode levar até dois meses.

Com o desenvolvimento desse trabalho, é pretendido que, com o fornecimento de novas informações, a reconstrução histórica do acidente seja realizada mais rapidamente, visando facilitar a identificação da causa do acidente e, então, agilizar o processo de perícia.

1.3 Objetivos do Projeto

1.3.1 Objetivo Geral

A proposta do trabalho é apresentar uma forma de adquirir dados que facilite a reconstituição histórica de acidentes de trânsito através da utilização de uma plataforma inercial e de um *software* que simule os movimentos do veículo durante a batida.

1.3.2 Objetivo Específico

Os objetivos específicos são:

- Conceber uma plataforma inercial que colete os dados necessários ao projeto;
- Desenvolver um software que simule os movimentos do carro em três dimenções durante a batida.

1.4 Resultados Obtidos

Os principais resultados obtidos foram:

- Criação de uma plataforma inercial que detecte um acidente de trânsito e armazene dados como aceleração, giro e posição do veículo antes e depois do acidente;
- Foi desenvolvido um *software* que simule acidentes de trânsito, usando os dados obtidos pela plataforma inercial e pós-processados pelo MATLAB.

Capítulo 2

Fundamentos

A seguir, serão abordados os diferentes processos de aquisição de dados por IMU's, a estrutura da plataforma inercial e o pós processamento dos dados coletados pelo software desenvolvido para a simulação 3D.

2.1 Sensores

2.1.1 Unidade de Medida Inercial - IMU

A Unidade de Medida Inercial (*Inertial Measurement Unit* -IMU) é um equipamento eletrônico que realiza medições da aceleração ao longos das direções longitudinais, verticais e laterais, além da taxa de rotação sobre os eixos X, Y e Z, como pode ser verificado na Fig. 2.1. Porém, isso só é possível com a integração de dois tipos de sensores: o giroscópio, que é um dispositivo que utiliza dos princípios de conservação do momento angular para estimar a orientação, e acelerômetro, que é um dispositivo capaz de medir acelerações causadas por forças específicas.

2.1.1.1 Microssistemas Eletromecânicos (MEMS)

Microssistemas Eletromecânicos, mais conhecidos pela sigla em inglês MEMS (*Microelectrical-mechanical Systems*), fazem parte de uma tecnologia que permite a criação de dispositivos eletromêcanicos (circuito eletrônico conectado a um circuito mecânico) em escala microscópica, muito utilizadas em microeletrônica [11]. Dentre as várias configurações existentes desses dispositivos pode-se destacar os microssensores inerciais: microacelerômetros e microgiroscópios [12].

O giroscópio é um dispositivo que permite medir a velocidade de rotação de um objeto em torno do centro de rotação; dito de outra forma, quão rápido o objeto gira em relação a um sistema de referência inercial, o que é conhecido como velocidade angular. A velocidade angular pode ser integrada em relação ao tempo para determinar a posição angular do objeto [13].

Um giroscópio mecânico geralmente consiste em um disco rotativo, onde os eixos que estão ligados nele são capazes de se mover livremente em qualquer orientação, como é possível verificar na Fig. 2.2.

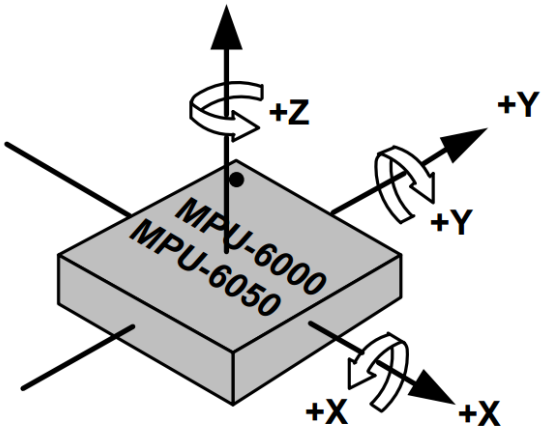


Figura 2.1: Ângulos e velocidades de uma unidade inercial (IMU) [1].

Um giroscópio microeletromecânico (MEMS) é muito parecido com o mecânico, porém ao invés de possuir um disco giratório, consiste em um tipo de ressonador vibrando. A ideia não muda, um objeto de vibração tende a continuar vibrando no mesmo plano que as suas bases de apoio.

De maneira geral, todos os giroscópios fabricados com tecnologia MEMS favorecem-se do efeito de Coriolis, baseado na transferência de energia entre dois modos principais de oscilação pela combinação de um movimento oscilatório de uma massa e a rotação do sistema que, pelo efeito de Coriolis, imprime na massa um movimento em direção ortogonal à da rotação [13].

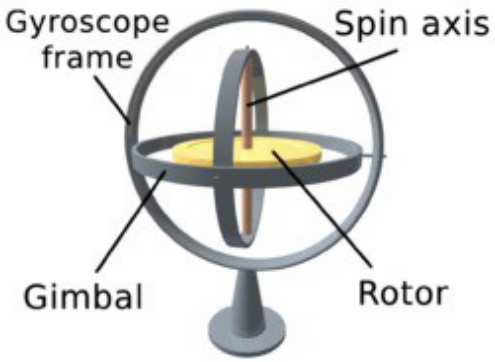


Figura 2.2: Giroscópio Mecânico.
Fonte: `https://pt.wikipedia.org/wiki/Giroscopio`

2.1.1.2 Microacelerômetro

Os acelerômetros são dispositivos que captam variações de velocidade e as transformam em um sinal elétrico. Existem vários tipos de acelerômetros, sendo que, dentro dos acelerômetros mecânicos, os mais comuns são: os capacitivos, os piezoelétricos e os piezoresistivos. Porém, recentemente, os acelerômetros mecânicos começaram a ser substituídos pelos acelerômetros eletromecânicos, como os MEMS.

Os acelerômetros MEMS podem medir a aceleração e a vibração em um, dois ou três eixos. O

sensor MEMS de um acelerômetro linear é baseado numa estrutura em silicone, com interdigitações e em forma de pente composta por dedos fixos e móveis. A aceleração é obtida da medição dos deslocamentos de elementos móveis que estão associados aos eixos. O movimento medido pelo sensor é então convertido num sinal analógico ou digital [14].

2.1.2 Tratamento dos dados

Na maioria dos casos, os dados do giroscópio são fundidos aos dados do acelerômetro. Por exemplo, usa-se um filtro complementar para minimizar o efeito de escorregamento do giroscópio [15]. No presente trabalho, a experiência demonstrou que o acelerômetro é muito sensível e ruidoso. É possível suavizar os dados através de fusão de sensores.

Tendo ambos os dispositivos (acelerômetro e giroscópio) na mesma plataforma facilita a correção de erros nas medidas, pelo fato de ambos estarem submetidos aos mesmos efeitos. Também há a possibilidade de utilizar a medida da aceleração linear, dada pelo acelerômetro, para a correção de rotações, obtidas pelo giroscópio.

2.1.3 GNSS

Um Sistema de Navegação Global por Satélite ou *Global Navigation Satellite System* (GNSS) é caracterizado por uma constelação de satélites artificiais com cobertura global, que emitem sinais continuamente, permitindo aos usuários encontrarem seu posicionamento nas três dimensões espaciais. Entre os sistemas de posicionamento, destacam-se o *Global Positioning System* (GPS) que se popularizou devido aos serviços oferecidos e por serem primeiro sistema global a ficar completamente em operação. Todavia, hoje o *GLobalnaya NAvigationnaya Sputnikovaya Sistema* (GLONASS) da Federação Russa, também se encontra em plena operação [16], já o *Galileo* ,da União Europeia, o *BeiDou*, da República Popular da China, e o *Quasi-Zenith Satellite System* (QZSS), do Japão, estão em fase de implementação [17].

Posicionar um objeto significa determinar as suas coordenadas em relação a um referencial específico, utilizando algum tipo de informação. No caso do GNSS, o posicionamento é feito por meio da técnica chamada de triangulação (ver Fig. 2.3), onde o objeto é um aparelho receptor GNSS [18].

Os princípios básicos do posicionamento GNSS envolvem resolver equações baseadas na triangulação, com os dados obtidos das constelações de satélites GNSS, que tem suas coordenadas conhecidas. A localização do receptor é determinada com base nos sinais GNSS transmitidos pelos satélites. [19].

Os satélites das constelações GNSS emitem sinais codificados, chamados de códigos pseudo aleatórios ou em inglês *pseudo random noise* (PRN), que os identificam. O aparelho receptor processa o sinal recebido e calcula a distância do receptor até o satélite que emitiu o sinal. Como o sinal emitido é uma onda eletromagnética, ela se propaga a uma velocidade de 299792458 m/s [20] (aproximadamente 300000 km/s), então basta saber o instante de tempo que o sinal foi emitido, o

instante de tempo de chegada do sinal e posição do satélite para saber a distância entre o satélite e o receptor. Porém, com somente a distância obtida por um satélite, o receptor poderá estar em qualquer região de uma superfície esférica, com o satélite no centro da esfera. Já com o sinal de dois satélites, o receptor poderá estar na região de intersecção de duas esferas, que é uma circunferência. Desse modo, somente com três satélites seria possível encontrar a posição do receptor [21], como ilustra a Fig. 2.3.

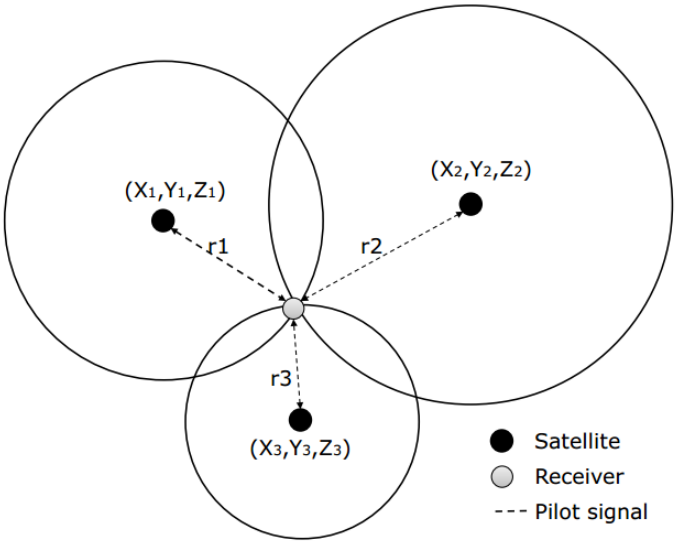


Figura 2.3: Técnica de triangulação.
Fonte: `http://www.howtechnologywork.com/how-does-a-gps-work/`

Na teoria, seriam necessários três satélites para determinar a longitude, latitude e altitude do satélite. Todavia, como as ondas de rádio emitidas pelos satélites são ondas eletromagnéticas, de velocidade muito alta, bastava um erro de sincronização dos relógios de apenas 1 μs, que o erro na posição do receptor seria de no mínimo 300 m. Por isso, um quarto satélite é necessário para calcular a posição do receptor com uma precisão aceitável, considerando o possível erro de sincronização dos relógios dos satélites e do receptor.

A acurácia do posicionamento depende de vários fatores, como: algoritmo utilizado, números de satélites, receptor utilizado, etc (ver Tabela 2.1 e Fig. 2.4). Minimizar os possíveis erros envolvendo o posicionamento, significar investir receptores GNSS que em geral, tem qualidade diretamente proporcional ao preço.

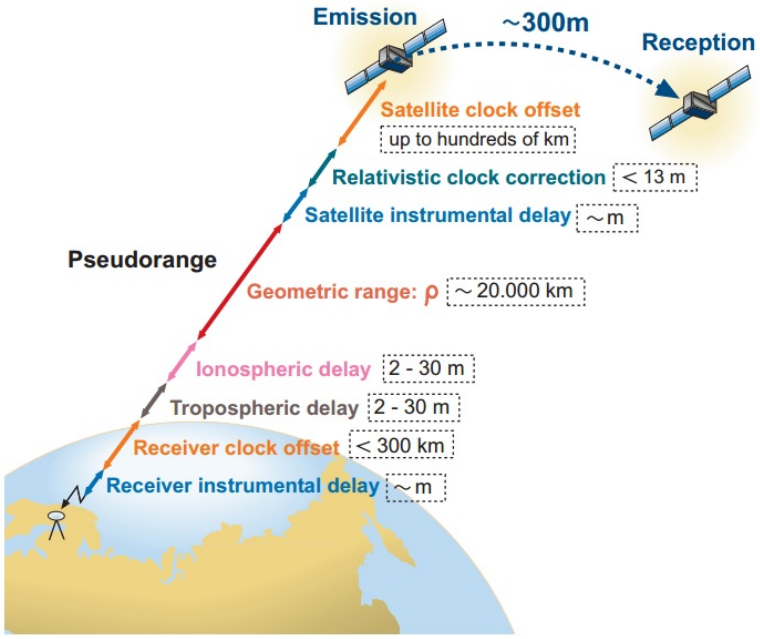


Figura 2.4: Diferentes erros envolvidos no posicionamento via satélite.

Tabela 2.1: Resumo dos tipos de erros no GNSS [2].

Fonte de erro	**Tipo de erro**
Satélite	Atraso entre as duas portadoras no hardware do satélite Erro de órbita Erro do relógio
Propagação do sinal	Refração troposférica Refração ionosférica Perdas de ciclos Multicaminho
Receptor/antena	Erro do relógio Erro entre os canais Variação do centro de fase da antena Fase *wind-up*
Estação	Erro nas coordenadas Multicaminho Marés terrestres Carga dos oceanos e atmosféricas

2.2 Formas de Comunicação

2.2.1 Serial

A porta serial envia e recebe bytes de informação um bit de cada vez e, apesar de ser mais lenta que a comunicação paralela, para uma mesma frequência, oferece simplicidade na conexão, pois envolve uma menor quantidade de fios. Além disso, a comunição serial é mais simples e pode ser utilizada em distâncias maiores.

A comunicação se torna completa com a utilização de 3 linhas de transmissão: (1) Terra, (2) Transmissão, e (3) Recepção. Em geral, é *full-duplex* e por isso, a porta está apta a transmitir dados em uma linha enquanto recebe dados em outra. As características importantes da serial são taxa de transmissão (*baud rate*), bits de dados (*data bits*), bits de parada (*stop bits*), e paridade.

2.2.2 I2C

I2C (*Inter-Intergrated Circuit*) é um barramento serial desenvolvido pela Philips, que é usado para conectar periféricos de baixa velocidade a uma placa mãe, a um sistema embarcado ou a um telefone celular.

O barramento I2C utiliza o modelo mestre-escravo (*master-slave*), que deve possuir pelo menos um dispositivo que atue como mestre e outros que atuem como escravos. A função do mestre é coordenar a comunicação, sendo ele quem realiza o envio e a consulta de informações sobre algum escravo.

Apenas duas linhas bidirecionais de dreno aberto, Dados Seriais (*Serial Data* - SDA) e *Clock* Serial (*Serial Clock* - SCL), são usadas pelo I2C. Este protocolo especifica dois sinais de comunicação, um com o sinal de *clock*, gerado pelo mestre e unidirecional, e outro de dados, bidirecional. Um esquema básico mostrando as conexões de um barramento I2C pode ser visto na Fig. 2.5.

O número de dispositivos escravos em um único barramento é limitado tanto pelo tamanho do endereço, que pode ser de 7 bits ou 10 bits, como também por restrição de espaço, já que não se pode ultrapassar poucos metros de fios, já que capacitância total máxima, de aproximadamente 400 pf, impede o funcionamento correto do barramento.

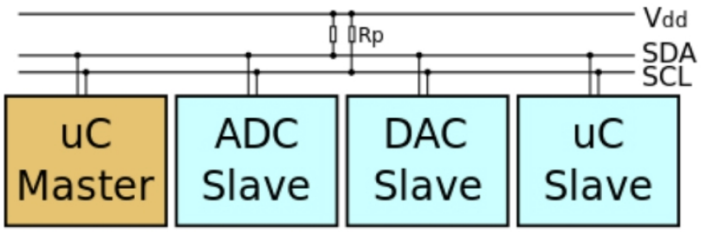


Figura 2.5: Esquema básico de ligação de um barramento I2C.
Fonte: `https://pt.wikipedia.org/wiki/I2C`

Quando utilizada a comunicação I2C, é importante saber o endereçamento de cada dispositivo,

já que todos são ligados no mesmo barramento. É por meio dos bits de endereço configurável que é possível definir quem irá receber os dados e garantir que um dispositivo errado não receba dados que seriam de outro dispositivo.

2.3 Memória EEPROM

Do inglês *Electrically Erasable Programmable Read-only Memory*, ou EAROM (*Electrically Alterable Read-only Memory*), a EEPROM é uma ROM programável pelo usuário, que pode ser apagada e reprogramada quantas vezes o usuário quiser através da aplicação de uma tensão mais elevada do que a tensão normal de operação. Porém,o apagamento e reprogramação tem de ser feito para todo o conteúdo da EEPROM, e não seletivamente.

Internamente a EEPROM é dividia em bytes, podendo cada byte ser acessado individualmente através de seu endereço que, na verdade, nada mais é do que a posição do byte na memória. A memória EEPROM usada nesse projeto é do modelo 24LC1025, da marca Microchip Technology Inc. Ela possui o modo de escrita byte e página, com capacidade máxima de 128 bytes por escrita. Já a leitura de dados, pode ser aletória ou sequencial. [22].

2.4 Simulador 3D

A simulação é um processo de projetar um modelo computacional de um sistema real com o propósito de entender seu comportamento.

De acordo com [23], os principais passos numa simulação visual 3D são: 1. criar objetos; 2. criar cenário; 3. aplicar textura aos objetos; 4. adicionar luz; 5. criar câmera; 6. animação e 7. renderização.

O processo de visualização da animação consiste basicamente na exibição de uma série de imagens, a certa velocidade, onde os objetos mudam suas posições no cenário a partir de parâmetros definidos. No caso deste trabalho, o interesse é em Simulação 3D.

2.4.1 Plataforma de Desenvolvimento Unity 3D

O Unity 3D se apresenta como um *Game Engine*, que é um programa de computador utilizado na confecção dos jogos digitais. É na *game engine* que toda a programação do jogo é feita e são unidos os arquivos de áudio, imagens e modelos 3D utilizados para criar os diversos cenários e ambientes do jogo.

Essa plataforma foi escolhida para a criação do simulador por possuir um estilo de programação e organização dos projetos muito simples. Além disso, traz a possibilidade de utilizar elementos criados por outras pessoas através de downloads realizados na sua loja oficial (Asset Store - `https://www.assetstore.unity3d.com/`).

Entre outras vantagens do Unity 3D, podemos citar:

- Disponibilização de ferramentas de aprendizado para o desenvolvedor - no site do Unity (`http://unity3d.com/learn`) é possível encontrar diversos tutoriais, além de toda a documentação necessária para o desenvolvedor utilizar as classes do Unity em seus scripts;
- Possui uma versão gratuita - que não contém os elementos avançados disponíveis na ferramenta, mas é uma excelente opção para jogos mais simples e;
- Permite o desenvolvimento de jogos pra diversas plataformas e não é necessário nenhuma programação extra, apenas a reconstrução do projeto com a plataforma-alvo selecionada.

2.5 Filtragem de sinais

2.5.1 Filtro de média

Um filtro de média (*average filter*) é definido por:

$$\overline{x}_k = \frac{x_1 + x_2 + ... + x_k}{k} \tag{2.1}$$

Tendo em conta a seguinte expressão:

$$\overline{x}_{k-1} = \frac{x_1 + x_2 + ... + x_{k-1}}{k-1}, \tag{2.2}$$

pode ser obtida uma forma recursiva do filtro de média, muito útil para o caso de dados sendo coletados por sensores num sistema embarcado:

$$\overline{x}_k = \frac{k-1}{k}\overline{x}_{k-1} + \frac{1}{k}x_k \tag{2.3}$$

Esta expressão mostra o cálculo da média para um novo dado, tendo em conta uma média anterior.

2.5.2 Filtro de média móvel

O filtro média móvel é muito empregado em processamento digital de sinais, principalmente por sua facilidade de uso [24]. O filtro de média móvel implementa a média de um número de dados k tomados recentemente. Neste caso, a média não tem em conta todas as medida tomadas durante o experimento. Somente os últimos n dados são objeto de estudo. Ou seja, de certa forma o filtro média móvel é uma evolução do filtro média. Matematicamente, ele pode ser dado da seguinte forma:

$$\overline{x}_k = \frac{x_{k-n+1} + x_{k-n+2} + ... + x_k}{n} \tag{2.4}$$

Novamente podemos definir:

$$\overline{x}_{k-1} = \frac{x_{k-n} + x_{k-n+1} + ... + x_{k-1}}{n}, \tag{2.5}$$

podendo-se gerar uma forma recursiva para este tipo de filtragem:

$$\overline{x}_k = \overline{x}_{k-1} + \frac{x_k - x_{k-n}}{n} \tag{2.6}$$

2.5.3 Filtro de Kalman

O filtro de Kalman é um filtro recursivo, usado para predizer os estados de um sistema dinâmico, muito usado em aplicações em tempo real [25]. Para fazer uso do filtro, o primeiro passo é modelar o sistema usando a notação de espaço de estados, dado pelas equações à seguir:

Descrição do sistema

$$\dot{x} = \mathbf{A}x + \mathbf{B}u \tag{2.7}$$

$$z = \mathbf{H}x, \tag{2.8}$$

onde x é o vetor de estados, z é a variável de medida e as matrizes $\mathbf{A}$, $\mathbf{B}$ e $\mathbf{H}$ são matrizes que caracterizam o sistema em questão, obtidas por modelagem.

Os estados e as medida do sistema, são considerados corrompidos por ruído branco gaussiano com média zero e covariância, de valores $\mathbf{P}$ e $\mathbf{Q}$, respectivamente. O filtro é composto por duas etapas seguintes, a etapa de predição e etapa de atualização, que se alternam. [26]. Matematicamente essas etapas são descritas da seguinte forma:

Predição

$$x_{k+1} = \mathbf{A}x + \mathbf{B}u \tag{2.9}$$

$$\mathbf{P}_{k+1} = \mathbf{A}\mathbf{P}_k\mathbf{A}^T + \mathbf{Q} \tag{2.10}$$

Atualização

$$\mathbf{K} = \mathbf{P}_k\mathbf{H}^T(\mathbf{H}\mathbf{P}_k\mathbf{H}^T + \mathbf{R})^{-1} \tag{2.11}$$

$$x_{k+1} = x_k + \mathbf{K}(y_k - \mathbf{H}x_k) \tag{2.12}$$

$$\mathbf{P}_{k+1} = (\mathbf{I} - \mathbf{K}\mathbf{H})\mathbf{P}_k, \tag{2.13}$$

onde $\mathbf{R}$ é a matriz de covariância do ruído da medida. Na etapa de atualização, o ganho de Kalman, $\mathbf{K}$ é o que minimiza o erro médio quadrático da estimação [25]. O sucesso no uso do filtro de Kalman depende principalmente da estimação das matrizes $\mathbf{P}$, $\mathbf{Q}$ e $\mathbf{R}$. Como muitas vezes as equações usadas no filtro de Kalman são consideradas complexas, usa-se o filtro somente unidimensionalmente, como será mostrado nó Capítulo 5, na filtragem dos dados de aceleração e giro obtidos por sensores.

2.5.4 Filtro Gaussiano

O filtro Gaussiano, pode ser usado como um filtro passa baixa, tendo um importante papel no processamento de imagens [27]. Na sua forma unidimensional e assumindo que a distribuição a ser analizado tem média zero, o filtro ter a seguinte forma:

$$G(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}}, \tag{2.14}$$

onde σ é o desvio padrão.

Para filtragem de sinais com ruído branco Gaussiano, o filtro Gaussiano faz um bom trabalho pois elimina esse tipo de ruído. Como a maioria dos sinais aparecem na sua forma primitiva com variáveis que evoluem no domínio do tempo, é necessário primeiro usar a transformada de Fourier para se utilizar o filtro Gaussiano. O trecho de código a seguir mostra essa técnica:

```
function [y_gauss]=gaussian_filter(y,r,sigma,funcSize)
% y = Funcao de entrada
% r = Frequencia de corte
% sigma = Desvio padrao
% funcSize = Tamanho da funcao

Y = fft(y,funcSize-1);                          % Transformada de Fourier
gauss = zeros(size(Y));
gauss(1:r+1) = exp(-(1:r+1).^ 2 / (2 * sigma ^ 2)); % Frequencias positivas
gauss(end-r+1:end) = fliplr(gauss(2:r+1));      % Frequencias negativas
y_gauss = ifft(Y.*gauss,funcSize-1);            % Transformada inversa de
   Fourier
end
```

Observe que o sinal de entrada estava no domínio do tempo e, por isso, foi necessário a utilização da Transformada de Fourier, que passou o sinal para o domínio da frequência, no qual foram preservadas as frequências positivas e negativas.

2.6 Ângulos de Euler

O movimento relativo entre dois sistemas de coordenadas ortogonais pode ser representado por três ângulos independentes, chamados de ângulos de Euler, que são bastante usados na análise da dinâmica de corpos rígidos. Suas principais aplicações se encontram nas áreas da robótica, aviação, mecânica celestial, etc [28].

Existem diversas definições para os ângulos de Euler, porém, a adotada será a mesma da Fig.2.1, onde, é usado um sistema de coordenadas fixo no corpo rígido, adota-se regra da mão direita e mudanças de orientações são descritas por rotações de guiagem ϕ (*yaw*, em inglês), arfagem θ (*pitch*,

em inglês) e rolamento ψ (*roll*, em inglês) em torno os eixos, X, Y e X, respectivamente. Com uma convenção pré-estabelecida, é possível definir as matrizes de rotação, que podem rotacionar um vetor em torno dos eixos X, Y e Z, respectivamente [29]:

$$\mathbf{R_x}(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{bmatrix} \tag{2.15}$$

$$\mathbf{R_y}(\theta) = \begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix} \tag{2.16}$$

$$\mathbf{R_z}(\psi) = \begin{bmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.17}$$

Como existem três matrizes de rotações, elas podem ser comutadas em 6 formas distintas, gerando matrizes de rotações compostas. Por exemplo, para uma rotação sobre os eixos X, Y e Z, a matriz de rotação correspondente $\mathbf{R_{xyz}}$ será:

$$\mathbf{R_{xyz}} = \mathbf{R_x}(\phi)\mathbf{R_y}(\theta)\mathbf{R_z}(\psi) \tag{2.18}$$

$$\mathbf{R_{xyz}} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{bmatrix} \begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix} \begin{bmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{R_{xyz}} = \begin{bmatrix} \cos\theta\cos\psi & \cos\theta\sin\psi & -\sin\theta \\ \cos\psi\sin\theta\sin\phi - \cos\phi\sin\psi & \cos\phi\cos\psi + \sin\theta\sin\phi\sin\psi & \cos\theta\sin\phi \\ \cos\phi\cos\psi\sin\theta + \sin\phi\sin\psi & \cos\phi\sin\theta\sin\psi - \cos\psi\sin\phi & \cos\theta\cos\phi \end{bmatrix} \tag{2.19}$$

Já para a sequência de rotação sobre os eixos Y, X e Z, respectivamente, a matriz de rotação correspondente seria

$$\mathbf{R_{yxz}} = \mathbf{R_y}(\phi)\mathbf{R_x}(\theta)\mathbf{R_z}(\psi) \tag{2.20}$$

$$\mathbf{R_{yxz}} = \begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{bmatrix} \begin{bmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{R_{yxz}} = \begin{bmatrix} \cos\psi\cos\theta - \sin\theta\sin\phi\sin\psi & \sin\psi\cos\theta + \sin\theta\sin\phi\cos\psi & -\sin\theta\cos\phi \\ -\cos\phi\sin\psi & \cos\phi\cos\psi & \sin\phi \\ \cos\theta\sin\phi\sin\psi + \sin\theta\cos\psi & -\cos\psi\cos\theta\sin\phi + \sin\psi\sin\theta & \cos\theta\cos\phi \end{bmatrix} \tag{2.21}$$

Fica claro observando as matrizes acima que para diferentes ordens de rotação, a matriz rotacional composta equivalente é diferente. Isso ocorre, pois, multiplicação de matrizes não é comutativa. Para ilustrar esse efeito visualmente, considere o objeto mostrado na Fig. 2.6.

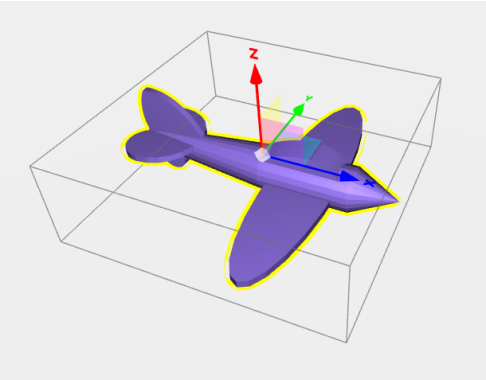


Figura 2.6: Avião em 3D.
Fonte: `https://free3d.com/3d-models/aircraft`

A Fig. 2.7 mostra a sequência de rotações de 90° em X, Y e Z, denominada $R_x(90°)$, $R_y(90°)$ e $R_z(90°)$. Já Fig. 2.8 mostra a sequência de rotações $R_y(90°)$, $R_x(90°)$ e $R_z(90°)$. É de se notar, que a configuração final do objetos para as duas sequências de rotações citadas é diferente.

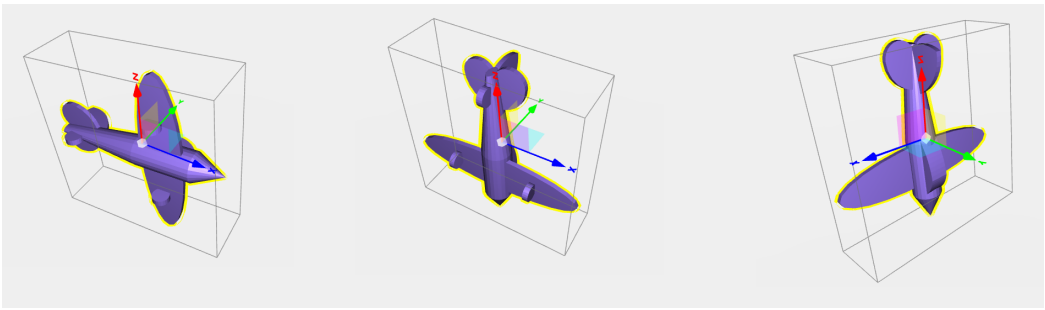


Figura 2.7: Sequência de rotações $R_x(90°)$, $R_y(90°)$ e $R_z(90°)$, mostradas da esquerda para direita.

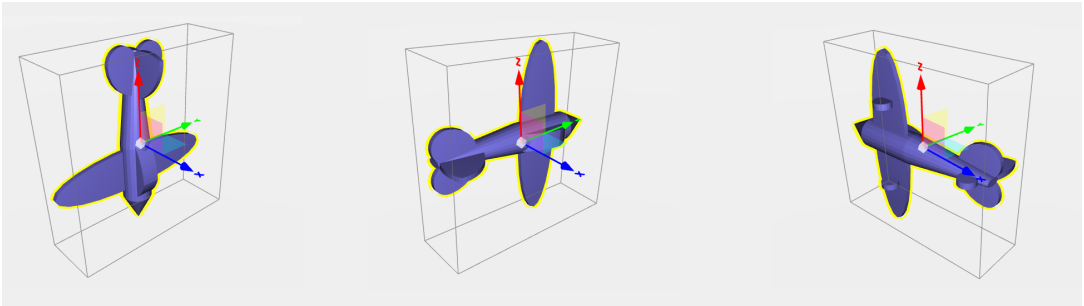
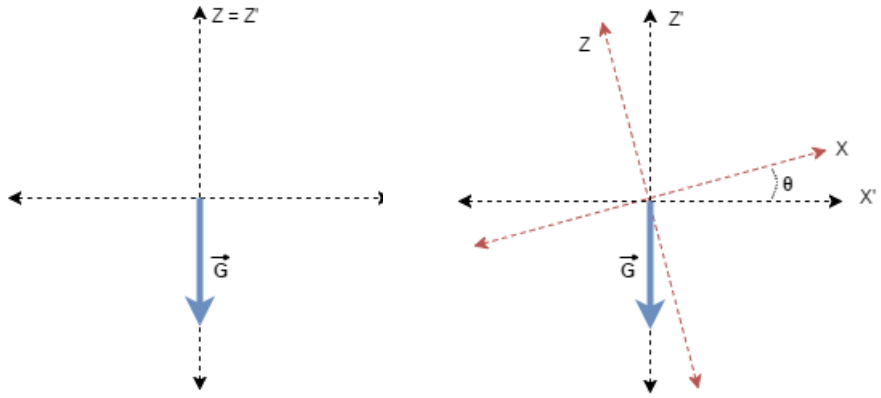


Figura 2.8: Sequência de rotações $R_y(90°)$, $R_x(90°)$ e $R_z(90°)$, mostradas da esquerda para direita.

Além da desvantagem de rotações usando ângulos de Euler não ser comutativas, outra consequência é que após algumas sequências de rotações, dois eixos podem ficar alinhados, gerando uma perda de grau de liberdade. Para contornar o problema, a alternativa é usar Quatérnios, que é uma expansão dos números complexos para três dimensões. Todavia, operações com quatérnios



(a) Calibração perfeita (b) Calibração incorreta

Figura 2.9: Influência da aceleração da gravidade sobre a calibração do acelerômetro

não são muito intuitivas, o que dificulta o sua utilização.

2.7 Efeito da aceleração da gravidade sobre o acelerômetro

Como citado anteriormente, o acelerômetro sempre é afetado pela aceleração gravitacional, o que para a aplicação de uma plataforma inercial para veículos, é indesejável. Eliminar de fato o efeito da aceleração gravitacional não é uma tarefa fácil, pois, diversos fatores influencial para que se possa obter bons resultados. Por exemplo, considerando um sistema de coordenadas $X'Y'Z'$, fixo à terra, e o sistemas de coordenadas do acelerômetro for XYZ, e que o acelerômetro não tenha sido calibrado corretamente, indicando valores como se o acelerômetro estivesse inclinado com um ângulo θ em relação ao eixo X', como ilustra a Fig. 2.9.

O vetor a aceleração da gravidade, em relação ao sistema de coordenadas XYZ, em *g's*, é dado por

$$\vec{\mathbf{G}} = \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix},$$

e com uma calibração o vetor aceleração medido pelo acelerômetro seria $\vec{\mathbf{A}}_1$=$\vec{\mathbf{G}}$. Porém, para a calibração errada, com uma inclinação θ=1° em relação ao eixo X, a aceleração $\vec{\mathbf{A}}_2$ medida pelo acelerômetro será

$$\vec{\mathbf{A}}_2 = \begin{bmatrix} g\cos\theta \\ 0 \\ -g\sin\theta \end{bmatrix} = \begin{bmatrix} g\cos 1^\circ \\ 0 \\ -g\sin 1^\circ \end{bmatrix} = \begin{bmatrix} 0,0175 \\ 0 \\ -0,9998 \end{bmatrix}$$

Com o acelerômetro parado em relação a Terra (sem rotação e translação), após 30 s e com o acelerômetro inicialmente na posição $\begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T$ e considerando g= 9.81 ms^{-2}, as posições $\vec{\mathbf{S}}_1$ e $\vec{\mathbf{S}}_2$ do acelerômetro considerando as acelerações $\vec{\mathbf{A}}_1$ e $\vec{\mathbf{A}}_2$, respectivamente serão:

$$\vec{\mathbf{S}}_1 = \frac{t^2}{2}\vec{\mathbf{A}}_1 = \frac{t^2}{2}\vec{\mathbf{G}} = \frac{30^2}{2}\begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix} \cdot 9.81 = \begin{bmatrix} 0 \\ 0 \\ -4414,5 \end{bmatrix} m$$

$$\vec{\mathbf{S}}_2 = \frac{t^2}{2}\vec{\mathbf{G}}_2 = \frac{30^2}{2}\begin{bmatrix} 0.0175 \\ 0 \\ -0.9998 \end{bmatrix} \cdot 9.81 = \begin{bmatrix} 77 \\ 0 \\ -4413,8 \end{bmatrix} m$$

Como os vetores de posição desejados seriam $\vec{\mathbf{S}}_1$=$\vec{\mathbf{S}}_2 = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T$, é possível notar o impacto negativo da aceleração da gravidade. Minimizar tais efeitos se chamar compensação de gravidade ou *gravity compensation*, em inglês. Uma das formas de fazer tal compensação seria, a partir do conhecimento inicial do vetor aceleração da gravidade, rotacioná-lo a cada nova leitura do acelerômetro. Os ângulos de rotação (ϕ, θ, ψ) poderiam ser obtidos pelo giroscópio. Tal metodologia é abordada na Capítulo 5.

Capítulo 3

Proposta de Emprego da Plataforma Inercial

Neste capítulo são apresentadas as principais características da proposta de plataforma inercial, incluindo uma descrição completa de toda a parte de hardware utilizada e de como foi feita a implementação de software, capaz de detectar e armazenar informações obtidas antes e depois do acidente de transito.

Com o objetivo de tornar a proposta de plataforma inercial vantajosa em relação aos módulos tradicionais, como os encontrados em módulos de *airbag*, por exemplo, o sistema necessitava ser projetado e desenvolvido para ser simples, barato e que garantisse uma boa resolução. Como mostrado na Fig. 3.1, a proposta consiste em uma plataforma inercial que obtivesse dados continuamente que permitissem caracterizar o movimento do veículo, antes e depois do acidente de trânsito. Em seguida, um *software* seria responsável pelo pós-processamento, no qual iria gerar um arquivo com informações específicas para a perícia técnica e também outro arquivo no formato texto para ser usado em um simulador 3D. Por enquanto, como a proposta ainda está em estudo, o pós-processamento está sendo feito pelo *MATLAB*.

Os componentes selecionados para a construção do *hardware* precisavam ser baratos, facilmente encontrados no mercado e de simples utilização. Por isso, os componentes da Tabela 3.1 foram selecionados. Desse modo, sem contar caixa de proteção para o *hardware*, o protótipo custaria somente R$ 140.

No trabalho já realizado anteriormente [8], o sistema já possuía um Arduino Mega 2560, um MPU-6050 e uma memória EEPROM 24LC1025. Porém, um dos problemas de um sistema com esses componentes é que a memória EEPROM, com a taxa de uso de algumas horas por dia, teria uma vida útil de menos de um ano, devido ao limite de escritas na memória EEPROM. Por isso, surgiu a necessidade de se adicionar uma memória SRAM, que tem uma vida útil extremamente elevada. Optou-se também, por adicionar um módulo GPS NEO6MV2, como outra forma de

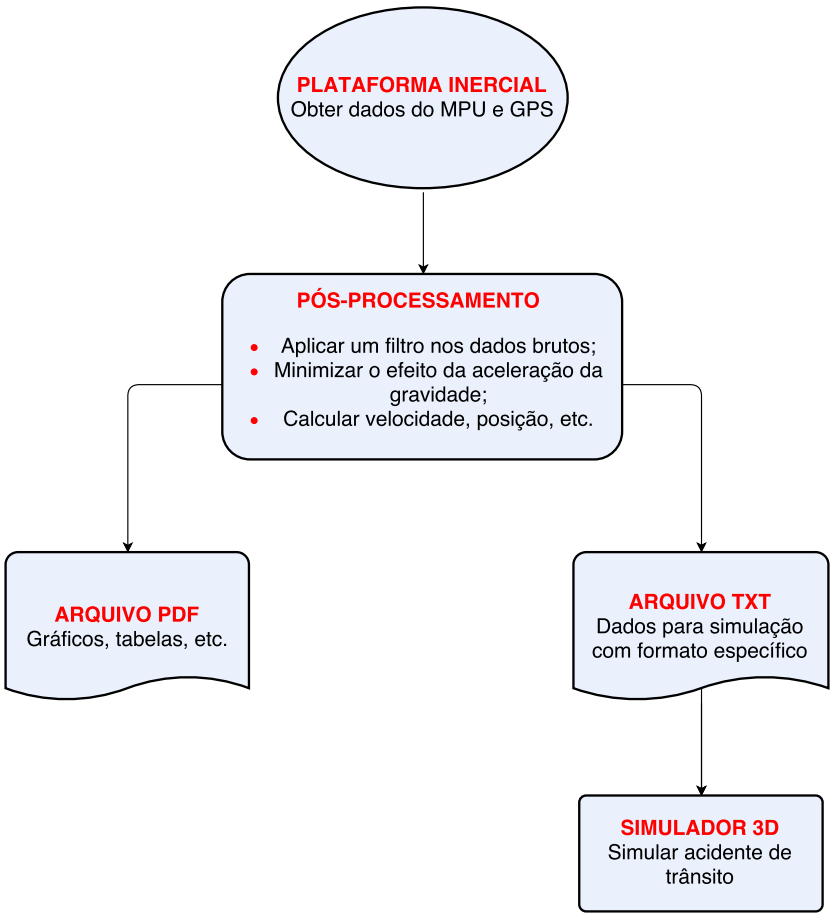


Figura 3.1: Resumo dos objetivos na criação da plataforma inercial.

Tabela 3.1: Principais componentes utilizados

Componente	**Quantidade**	**Preço médio (reais)**
Arduino Mega 2560	1	50
MPU-6050 (Acelerômero e giroscópio)	1	12
Memória EEPROM 24LC1025	1	9
Memória SRAM 23LC1024	1	9
Módulo GPS NEO6MV2	1	60
	TOTAL	**140**

obtenção de dados, como posição e velocidade do veículo.

Para o processamento, o Arduino Mega 2560 manteve-se o [30] por ter uma capacidade de processamento desejada e alto número de portas de entrada e saída analógicas/digitais. Sua alimentação pode ser entre 7 e 12V, que pode ser fornecida por uma grande parte dos veículos existentes [1]. Já para os periféricos, para coleta de aceleração e giro do veículo continuou sendo o módulo acelerômetro e giroscópio MPU-6050, pois é um componente bastante utilizado na eletrônica e possuí uma vasta gama de bibliotecas disponíveis na internet. Como uma forma secundária de colota de dados, pois como mencionado, nem sempre é possível obter um sinal de GPS com

qualidade, foi escolhido o módulo GPS NEO6MV2 [31], uma vez que, em relação a outros módulos encontrados no mercado, esse é relativamente barato e de fácil utilização.

Para a gravação de dados, caso não ocorresse um acidente de trânsito, foi necessário uma memória volátil, de alta velocidade de escrita de dados e com capacidade que atendesse os critérios de projeto. Caso ocorresse um acidente, os dados seriam copiados da memória volátil para uma não volátil, com o objetivo de garantir que os dados não fossem perdidos. Como os dados de aceleração e giro nos três eixos (X, Y e Z) obtidos a cada ciclo pelo MPU-6050 eram de 12 bytes, para uma obter dados de aproximadamente 15 s antes e 15 s após o acidente a um período de amostragem de 5 ms, por exemplo, seria necessário no mínimo 72 kbytes de memória. Por isso, foram escolhidas as memória EEPROM [22] e SRAM 23LC1024 [32], ambas de 128 kbytes de armazenamento. A comunicação entre os componentes citados anteriormente é mostrada no circuito esquemático da Fig. 3.2 e na implementação da Fig. 3.3, que tem dimensões de aproximadamente 3 x 12 x 12 cm.

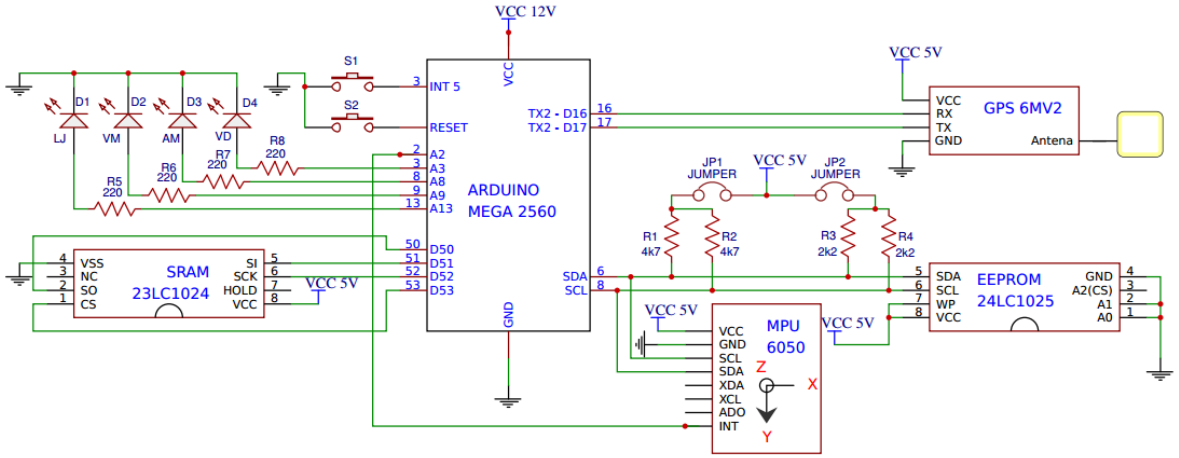


Figura 3.2: Esquema de comunicação.

A comunicação entre os componentes é feita de três formas diferentes: barramento I2C, barramento SPI e comunicação serial. Para o barramento I2C (pinos SDA e SDL), o Arduino Mega é o mestre e o MPU, junto a memória EEPROM, são os escravos. Para correto funcionamento do protocolo, são necessários dois resistores *pull-ups* e que nesse caso, de acordo com o *datasheets* dos componentes, eles podem ser de 4,7 kΩ ou 2,2 kΩ. Cada opção de *pull-up* oferece uma velocidade de barramento, e podem ser selecionada pelos *jumpers* da Fig. 3.2. Os pinos XDA e XCL do MPU não serão utilizados nesse trabalho, porém eles formam o barramento I2C auxiliar para a conexão de um outro dispositivo, como um magnetômetro.

No caso do barramento SPI, o Arduino Mega também é mestre e o único escravo é a memória SRAM. Por fim, o módulo GPS se comunica via porta serial, e faz uso de uma das três portas seriais do Arduino Mega. É importante notar que, a entrada serial RX do GPS se comunica com a TX do Arduino e a TX do GPS se comunica com a RX do Arduino.

Ainda de acordo com as Figuras 3.2 e 3.3, o sistema possuí quatro LED's (D1, D2, D3 e D4) e dois botões (S1 e S2). O LED verde (D4) fica aceso quando sistema está operando, amarelo (D3) quando algum erro for detectado, o vermelho (D2) para indicar se houve colisão e laranjado (D1)

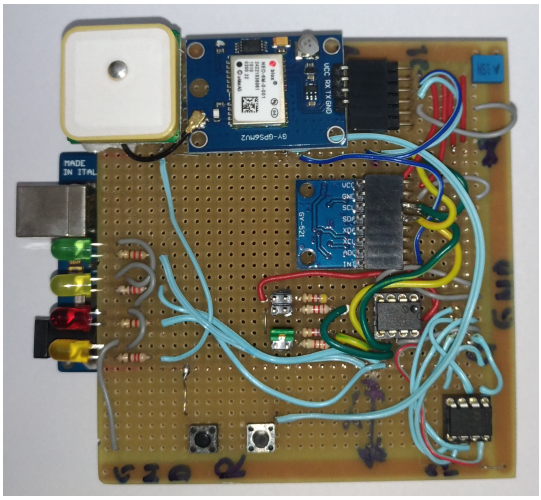


Figura 3.3: Protótipo de implementação da plataforma inercial.

acente quando existe sinal de GPS acessível. Esse último LED foi importante para a fase de testes, pois, nem sempre era possível obter sinal de GPS, principalmente em ambientes fechados. O botão S1 é o botão RESET, que reinicia o processador e enquanto o dispositivo está em fase de testes e o botão S2 serve para forçar uma batida, na qual os dados somente começam a ser gravados na memória quando o usurário pressiona o botão. Tal botão foi essencial para os testes em campo mostrados na Secção 5.

Para cada leitura completa do MPU, são obtidos dados de aceleração e giro nos três eixos (X, Y e Z) sobre seu sistema de coordenadas fixo, totalizando 12 bytes de dados. Já para o módulo GPS, diversas informações podem ser obtidas, como latitude, longitude, altitude, idade da informação, etc.

3.1 Mapeamento das memórias

Conforme mencionado anteriormente, para garantir a integridade dos dados obtidos pela plataforma inercial, caso a alimentação fosse perdida após o acidente de trânsito, seria necessário uma memória não-volátil e, consequentemente, selecionou-se uma memória EEPROM de 128 kbytes. Como a memória EEPROM possuí um limite ciclos de apagamento e gravação, para o armazenamento de dados antes do acidente, optou-se por utilizar uma memória volátil SRAM, de 128 kbytes. Desse modo, os dados da colisão somente seriam transferidos da memória SRAM para caso ocorresse uma colisão. Já que a memória EEPROM é o destino final dos dados após o acidente, sua capacidade de 128 kbytes, será considerada o limite para o mapeamento de memória.

Como os dados de aceleração e giro nos três eixos (X, Y e Z) obtidos a cada ciclo pelo MPU eram de 12 bytes, os dados obtidos pelo GPS (latitude, longitude, altitude, data e horário) totalizavam 24 bytes por ciclo de gravação e cada página de gravação da memória EEPROM dispõe de 128 bytes, optou-se por mapear a memória em quantidades de bytes que fosse o mínimo múltiplo comum (mmc) desses números, ou seja, 384. Experimentalmente, verificou-se que em média, independente

do GPS estar em funcionamento, o tempo para cada leitura completa do MPU e armazenamento da memória SRAM era de 1.8 ms.

Apesar do GPS constantemente fazer varreduras à procura de dados, de acordo com seu manual [31] e também constatado experimentalmente, o tempo para o módulo GPS 6MV2 adquirir sinais da constelação de satélites e dados de navegação e assim, calcular a posição do receptor, foi de aproximadamente 1 s. Esse tempo é conhecido como *Time to First Fix (TTFF)* para o caso *Hot Start*. No pior dos casos, quando o GPS acaba de ser ligado e se encontrar em um local com sinal de recepção ruim, em media o TTFF era de 27 s, também chamado de TTFF para *Cold Start*.

Tendo como base os tempos de ciclo para coleta de dados do MPU e GPS citados anteriormente, foi possível alocar espaço na memória EEPROM para armazenamento destes dados. Mantendo-se a página de configuração do sistema [8], de 128 bytes, a memória EEPROM pode ser mapeada da forma mostrada na Tabela 3.2.

Tabela 3.2: Disposição da pagina de configuração e dos dados do MPU e GPS na memória EEPROM

Endereço		Descrição	Tamanha de cada bloco (bytes)	Quantidade de blocos de dados
Inicial	Final			
0	127	Página de configuração	-	-
256	124671	Dados do MPU	12	10368
124672	130432	Dados do GPS	24	240

Desse modo, mesmo optando-se por usar ou não o GPS, com o tempo para cada leitura completa do MPU e armazenamento da memória SRAM sendo de 1,8 ms, o sistema poderia armazenar 10368$\cdot$ 0,0018 = 18,66 s de informação, ou seja, aproximadamente 9 s antes e 9 s após a colisão. Tal período de amostragem está bem inferior a de muitos módulos atuais no mercado, o que proporciona mais acurácia para o pós-processamento dos dados. É claro que, com o sistema funcionando continuamente e sem a detecção de colisão, os dados são armazenados na memória SRAM sobrescrevendo dados mais antigos.

A identificação de um acidente é dada quando algum dos limites de aceleração ou giro em algum dos três eixos (X, Y e Z) é atingido. Quando isso ocorre, armazena-se na página de configuração do sistema um valor do tipo *long*, indicando na memória EEPROM o momento exato que ocorreu a colisão. Outra forma de identificar quando ocorreu o acidente, é após a leitura dos dados obtidos pelo MPU, procurar pela linha de zeros, que é inserida como prenúncio das informações de um acidente.

3.2 Página de configuração do sistema

A página de configuração, que é o primeiro bloco de 128 bytes da memória EEPROM, foi reservada para manter informações importantes da plataforma inercial, que caracterize parâmetros do MPU e do GPS. Sua padronização foi praticamente a mesma adotada na referência [8]. As

principais mudanças foram a adição de informações referentes ao GPS e padronização para o português de algumas variáveis, como mostrado na Tabela 3.3. Entre as diversas informações apresentadas, é informado ao usuário se houve batida (endereço 0), *offset* obtido na calibração (endereço 4 ao 15), etc. É importante observar que o endereço 49 ao 127 da página não contêm informações e por isso, há espaço para dados adicionais que podem ser inseridos, se necessário. Por enquanto, tais posições se encontram vazias, gravadas com 0xFF.

Tabela 3.3: Parâmetros da página de configuração (Página 0) do protótipo

Endereço (bytes)	**Tipo de variável**	**Descrição**
0	Char	Pronta para uso? S = Sim e N = Não
1	Char	Passou no self test? S = Sim e N = Não
2	Byte	Escala do acelerômetro na calibração
3	Byte	Escala do giroscópio na calibração
4 e 5	Int	Offset do acelerômetro eixo X
6 e 7	Int	Offset do acelerômetro eixo Y
8 e 9	Int	Offset do acelerômetro eixo Z
10 e 11	Int	Offset do giroscópio eixo X
12 e 13	Int	Offset do giroscópio eixo Y
14 e 15	Int	Offset do giroscópio eixo Z
16	Byte	Escala do acelerômetro na operação
17	Byte	Escala do giroscópio na operação
18 e 19	Int	Limiar de disparo, eixo X, acelerômetro
20 e 21	Int	Limiar de disparo, eixo Y, acelerômetro
22 e 23	Int	Limiar de disparo, eixo Z, acelerômetro
24 e 25	Int	Limiar de disparo, eixo X, giroscópio
26 e 27	Int	Limiar de disparo, eixo Y, giroscópio
28 e 29	Int	Limiar de disparo, eixo Z, giroscópio
30 ao 33	Long	Linha de disparo do MPU
34	Char	Disparo no eixo X no acelerômetro? S = Sim e N = Não
35	Char	Disparo no eixo Y no acelerômetro? S = Sim e N = Não
36	Char	Disparo no eixo Z no acelerômetro? S = Sim e N = Não
37	Char	Disparo no eixo X no giroscópio? S = Sim e N = Não
38	Char	Disparo no eixo Y no giroscópio? S = Sim e N = Não
39	Char	Disparo no eixo Z no giroscópio? S = Sim e N = Não
40 e 41	Int	Ano
42	Byte	Mês
43	Byte	Dia
44	Byte	Hora
45	Byte	Minuto
46	Byte	Segundo
47	Byte	Milésimo de segundo
48	Char	Linha de disparo do GPS
49 ao 127	Char	Escrito 0xFF

3.3 Comunicação com o usuário

O usuário da plataforma inercial tem a opção de poder se comunicar com o dispositivo via porta serial USB e assim, executar algum dos comandos da Tabela 3.4. Por exemplo, é possível calibrar o dispositivo e obter os *offsets* das acelerações e giro nos três eixos, mostrar os dados em

tempo real, obter dados referentes à batida e zerar a memória EEPROM.

Tabela 3.4: Comandos para comunicação com o usuário

Comando	Descrição
0	Mostrar página de configuração
C ou c	Calibrar MPU
L ou l	Mostrar dados da EEPROM
R ou r	Mostrar dados em tempo real
S ou s	Mostrar dados da SRAM
Z ou z	Zerar memória EEPROM
Qualquer outra tecla	Parar de exibir informações

A Fig. 3.4 mostra um exemplo de página de configuração obtida ao enviar o caractere "0"pela porta serial. Já a Fig. 3.5, mostra um exemplo de dados brutos do MPU e GPS obtidos pelo dispositivo. Para os dados do MPU, a ordem de apresentação dos dados da esquerda para direita é: aceleração nos três eixos (X, Y e Z) e giro nos três eixos (X, Y e Z). Já para os dados do GPS, segue a seguinte ordem, da esquerda para a direita: ano, mês, dia, hora, minuto, segundo, milésimo de segundo, latitude (graus), longitude (graus), altitude (cm) e velocidade (km/s).

```
--> Início da primeira página
Data (GMT): 17/6/2017
Horario (GMT): 23:18:54:0
Pronta para uso? S
Passou no self test? S
Escala do acelerômetro na calibração: +/- 2g
Escala do giroscópio na calibração: +/- 250 graus/s
Offset do acelerômetro: X=-1630 Y=00191 Z=02574
Offset do giroscópio:   X=00719 Y=00170 Z=-0024
Escala do acelerômetro na operação: +/- 2 g
Escala do giroscópio na operação: +/- 250 graus/s
Limiar de disparo do acelerômetro: X=2 Y=2 Z=2 g
Limiar de disparo do giroscópio: X=500 Y=500 Z=500 graus/s
Linha de disparo: 1312
Disparo no acelerômetro? X=N Y=N Z=S
Disparo no no giroscópio? X=N Y=N Z=N
Linhha de disparo no GPS: 129904
--> Fim da primeira página
```

Figura 3.4: Exemplo de página de configuração (Página 0).

```
002016 -00476 -18644 -00628 -00148 000016
001928 -00536 -18652 -00615 -00166 000019
002004 -00564 -18668 -00599 -00168 000021
002060 -00496 -18624 -00609 -00175 000017
001956 -00520 -18708 -00635 -00160 000033
002036 -00508 -18676 -00601 -00175 000015
002088 -00532 -18596 -00611 -00172 000033
001996 -00488 -18596 -00596 -00172 000008
002140 -00440 -18692 -00621 -00172 000030
002028 -00468 -18644 -00631 -00162 000036
002124 -00428 -18644 -00662 -00179 000033
002040 -00516 -18704 -00638 -00145 000008
001988 -00436 -18664 -00670 -00157 000021
2017 06 17 23 18 36 00 -15.807721 -48.024620 1151.90 0.13
2017 06 17 23 18 37 00 -15.807721 -48.024620 1151.90 0.11
2017 06 17 23 18 37 00 -15.807721 -48.024620 1151.90 0.11
2017 06 17 23 18 38 00 -15.807721 -48.024620 1151.90 0.31
2017 06 17 23 18 38 00 -15.807721 -48.024620 1151.90 0.31
2017 06 17 23 18 39 00 -15.807721 -48.024620 1152.30 0.17
2017 06 17 23 18 40 00 -15.807721 -48.024620 1152.30 0.06
2017 06 17 23 18 40 00 -15.807721 -48.024620 1152.40 0.06
2017 06 17 23 18 41 00 -15.807721 -48.024620 1152.60 0.04
2017 06 17 23 18 43 00 -15.807721 -48.024608 1153.00 0.37
2017 06 17 23 18 43 00 -15.807721 -48.024608 1153.00 0.37
2017 06 17 23 18 44 00 -15.807721 -48.024608 1153.00 0.15
```

Figura 3.5: Leitura dos dados do equipamento após colisão.

3.4 Implementação do *software* no Arduino Mega

Utilizou-se o ambidente de deselvolvimento integrado (*Integrated Development Environment* - IDE) do Arduino, que adota a linguagem C/C++ para a criação do código, que foi carregado no Arduino Mega 2560 e que é responsável pela coleta, pré-processamento e armazenamento dos dados. De forma a facilitar a continuidade desse trabalho, o código foi implementado de forma simples, como mostrado nos Apêndices I, II, III e IV, usando técnicas de programação convencionais, de modo que o código pudesse ser facilmente alterado para melhorias.

Quatro bibliotecas foram necessárias para a criação do código: *Wire*, *SPI*, *SpiRam_Extended* e *TinyGPS*. A biblioteca *Wire*, que já é biblioteca padrão do Arduino, é responsável pela comunicação I2C e limita a transferência de dados entre mestre e escravo para no máximo 32 bytes, incluindo o endereço do destino. A biblioteca SPI, que também é nativa, é usada pela outra biblioteca *SpiRam_Extended*, que possibilita o uso da memória SRAM. A biblioteca *TinyGPS* é usada para comunicação com o módulo GPS, e assim como a *SpiRam_Extended*, foram escolhidas por terem funcionamento simples. Infelizmente, as bibliotecas *SpiRam_Extended* e *TinyGPS* não são nativas do Arduino e por isso, precisam ser baixadas e instaladas na IDE do Arduino. Tais bibliotecas podem ser facilmente encontradas no *site* `https://github.com/`.

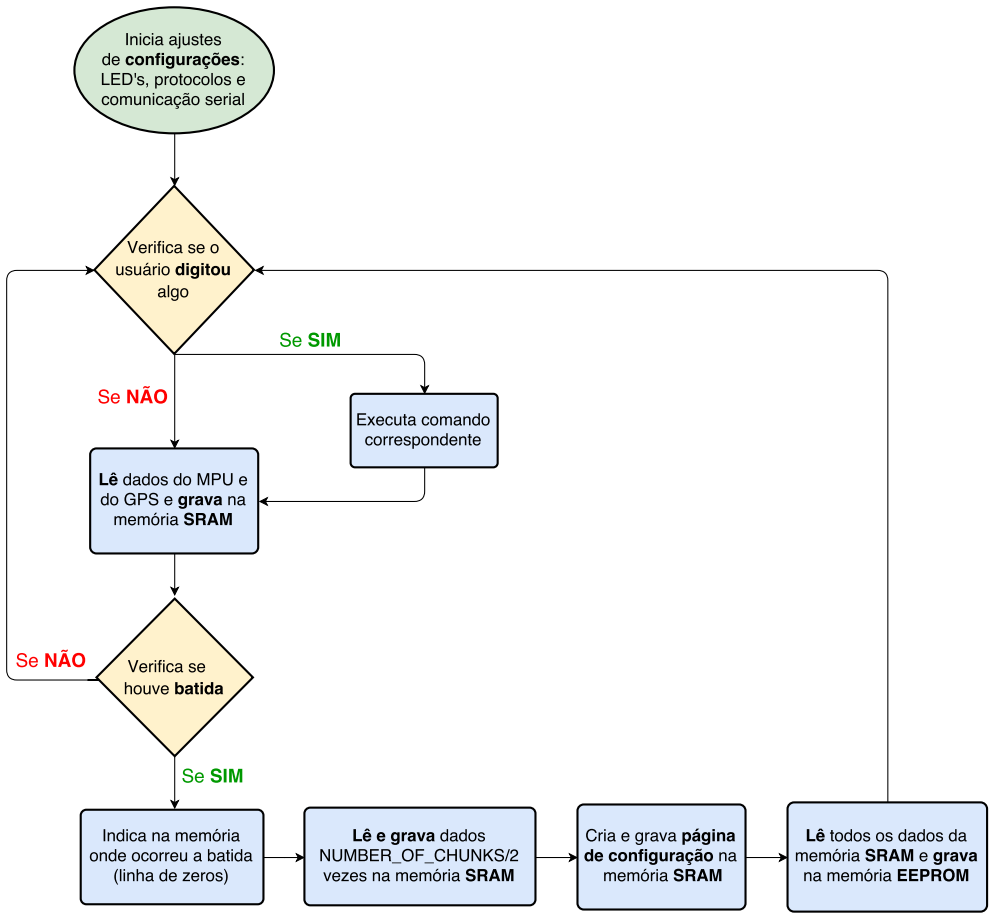


Figura 3.6: Resumo do funcionamento do código.

O funcionamento do código pode ser simplificado pelo diagrama da Fig. 3.6. Antes de executar o laço principal do programa, são feitas as configurações que serão usadas no decorrer do programa, como: definição das saídas para uso dos LED's, inicialização da comunicação serial com o monitor do IDE a 115200 bps, inicialização da comunicação serial a 9600 bps com o GPS e por fim, inicialização dos dispositivos responsáveis pelos dois protocolos, I2C e SPI, utilizados.

Em seguida, o programa entra em seu laço principal, na qual o primeiro passo a ser executado é verificar se o usuário digitou algum comando, que é caracterizado por algum caractere, como já mostrado na Tabela 3.4. Caso nenhum comando tenha sido digitado, em seguida são lidos os 12 bytes de dados do MPU e se disponível, os 24 bytes de dados do GPS, que posteriormente são armazenados na memória SRAM. O próximo passo é verificar se houve uma colisão, na qual os dados do MPU obtidos no passo anterior são comparados com os limites de aceleração e giro pré estabelecidos.

Caso seja detectado um acidente, o programa executa um laço de repetição continuamente até que todos os blocos de dados alocados para MPU, indicados pela Tab. 3.2, sejam ocupados por dados de aceleração e giro. O interessante é que, o código foi montado de tal forma que, é possível mapear a memória de diferentes formas só alterando algumas constantes do código apresentado no Apêndice I, na parte "*Memory Distribution*". Até mesmo se fosse necessário

expandir a capacidade de armazenamento das memórias EEPROM e SRAM, isso seria possível modificando essas constantes.

Por fim, o programa cria a página de configuração da Tabela 3.4 e lê todos os dados da memória SRAM e armazena na memória EEPROM. Esse último processo leva cerca de 45 s. Por enquanto, o programa retorna ao laço de repetição principal, porém, para uma aplicação real seria interessante que mesmo após outra colisão ser detectada, os dados da memória EEPROM só pudessem ser sobreescrevidos após um determinado período de tempo (por exemplo, de três dias), para garantir que o usuário realmente não perca esses dados.

Capítulo 4

Proposta de Simulador 3D

Neste capítulo é explicado como foi contruído o simulador 3D que, com os dados coletados pela plataforma inercial desenvolvida nesse projeto, é capaz de reproduzir os movimentos realizados pelo carro durante uma batida. Além disso, é explicado todo o seu funcionamento e suas limitações.

4.1 Inicialização

Ao ser inicializado, o simulador executa a leitura do arquivo texto que é gerado pelo Matlab e separa os dados em grupos através de vetores. Ao todo, 12 vetores são preenchidos após a inicialização do simulador e são responsáveis por armazenar todas as informações utilizadas na simulação. Dessa forma, caso haja a necessidade de repetir a simulação, não é preciso realizar uma nova leitura do arquivo texto, dado que todas as variáveis já estão salvas no próprio simulador - o que economiza tempo e processamento.

4.2 Câmeras

A tela inicial do simulador pode ser verificada na Fig.4.1. Do lado direito da tela consta um menu que contém as imagens do conjunto de câmeras que permitirão a visualização da simulação em diferentes ângulos, de forma a possibilitar uma avaliação sob diferentes perspectivas. Foram definidas 5 câmeras, em que:

- Câmera 1: é fixa e permite a visualização da parte traseira do carro (ver Fig 4.1);
- Câmera 2: é orbital e permite visualizar o carro em diferentes ângulos e distâncias, e é comandada pelo mouse (ver Fig.4.2a);

Figura 4.1: Tela inicial do simulador - câmera 1.

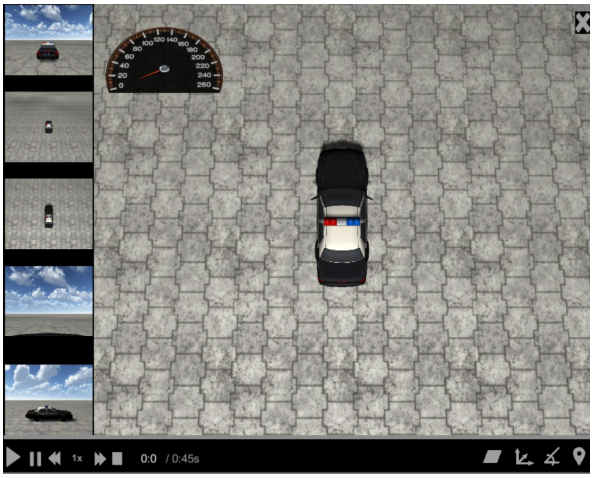
- Câmera 3: é fixa e apresenta a vista superior do carro (ver Fig.4.2b);
- Câmera 4: é fixa e tem como objetivo representar a visão do condutor do carro. Também é chamada de câmera em primeira pessoa (ver Fig.4.2c);
- Câmera 5: é fixa e permite observar a lateral direita do carro, (ver Fig.4.2d).

Inicialmente, a câmera 1 é definida como a principal e tem sua visão expandida para o resto da tela. Porém, é possível definir qualquer uma das outras câmeras como a principal, bastando clicar na visão da câmera escolhida no menu.

4.3 Botões de Controle

À esquerda do menu localizado na parte inferior da tela, encontram-se os botões responsáveis por controlar a simulação. São eles: o botão "iniciar", "pausar", "desacelerar", "acelerar"e "interromper". Entre os botões de desacelerar e acelerar, é encontrada a *label* responsável por informar a velocidade atual da simulação, sendo "1x"a velocidade em tempo real. Nesse simulador foram incorporadas as seguintes velocidades: 0.25x, 0.5x, 1x, 2x e 4x.

(a) Câmera 2 - Orbital



(b) Câmera 3 - Superior

(c) Câmera 4 - Primeira pessoa

(d) Câmera 5 - Lateral

Figura 4.2: Diferentes câmeras de visualização para o Simulador

Após os botões de controle, constam duas variáveis que são responsáveis por indicar o tempo atual da simulação e sua duração. O tempo total de simulação é sempre calculado considerando o tempo real, ou seja, na velocidade 1x. A barra de progresso, situada logo acima, foi implementada para proporcionar uma nova perspectiva do tempo de duração.

4.4 Botões Adicionais

Localizados à direita dos botões de controle, se encontram os botões adicionais. Esses botões tem como objetivo ativar ou desativar funcionalidades durante a simulação. Cada um desses botões é explicado nos tópicos abaixo.

4.4.1 Plano de Sustentação do Veículo

É o primeiro botão, representado por um paralelogramo, e é responsável por tornar visível ou invisível o plano de sustentação do veículo, Fig. 4.3. Essa funcionalidade foi implementada devido à uma limitação do simulador que será explicado posteriormente.

(a) Plano de sustentação visível

(b) Plano de sustentação invisível

Figura 4.3: Tela do simulador quando o plano de sustentação do veículo está visível e invisível.

4.4.2 Acelerações Gravitacionais

O segundo botão, da direita para a esquerda, quando ativado, torna visível a tabela e os eixos de acelerações da gravidade, medidas em g (ver Fig.4.4). Por meio desse conjunto, é possível verificar qual é a intensidade da aceleração aplicado no carro nos eixos X, Y e Z.

Para tornar visual a relação da aceleração segundos os eixos, e para que a tela de exibição da simulação não ficasse tão poluída, optou-se em tornar o tamanho dos eixos variável de acordo com o seu valor em g. A faixa de valores aceitáveis é de -3 g até 3 g. Se a medida estiver fora dessa faixa, o eixo irá apresentar seu tamanho mínimo ou máximo, dependendo do caso, e o valor real será mostrado na tabela de acelerações gravitacionais.

4.4.3 Ângulos

Ao ativar o terceiro botão da direita para a esquerda, é apresentado na tela uma seta na cor cinza que representa a posição angular inicial do carro no eixo Y. Ao longo do tempo, quando o carro começa a se locomover, uma nova seta, agora na cor azul, aparece na tela para representar a posições angular atual do carro, também para o eixo Y. Dessa forma, é possível comparar a posição angular inicial em relação a final do veículo para esse eixo de forma mais prática e visual, como pode ser observado na Fig.4.5.

O botão de ângulos também habilita a visualização de uma tabela em que são informados os valores atuais do ângulo do carro em cada um dos eixos.

Figura 4.4: Tela do simulador com tabela e eixos de acelerações da gravidade.

4.4.4 GPS

Habilitando o botão GPS, localizado na extrema direita, uma pequena janela é aberta na tela de simulação, Fig. 4.6. Nessa janela é possível visualizar no Google Maps a posição que o carro se encontra naquele instante e o trajeto realizado ao longo do percurso.

4.5 Velocímetro

Na lateral esquerda superior foi implementado um velocímetro para visualização da velocidade. A escolha do velocímetro analógico ao invés do digital foi baseado em dois motivos. O primeiro motivo é que ao utilizar um velocímetro analógico todas as medidas de velocidade coletadas podem ser utilizadas. No caso de um velocímetro digital, grande parte das medições poderiam não ser compreendidas, dado que a taxa de atualização do simulador é muito alta e tornaria quase impossível a visualização dos valores.

O segundo motivo é relativo ao fato de que o velocímetro analógico possui uma visualização mais intuitiva. Isso porque somente de saber onde o ponteiro se encontra, desde que a escala esteja

Figura 4.5: Comparação das posições angulares iniciais e finais do veículo.

adequada, é possível ter uma noção da velocidade.

4.6 Limitações do Simulador

4.6.1 Determinação da Taxa de Renderização

A plataforma Unity 3D possui a configuração *targetFrameRate*, que instrui o programa em construção a tentar renderizar em uma taxa de quadros determinada. Porém essa configuração não garante que o valor especificado seja alcançado, dado que podem ocorrer flutuações devido à especificações da plataforma ou mesmo porque o computador pode ser muito lento para conseguir.

Após a realização de vários testes, foi constatado que o computador utilizado no desenvolvimento desse projeto consegue executar os comandos solicitados a uma taxa máxima de 40 fps. Essa taxa representa atualizações a cada 25 ms, que é bem superior a taxa de amostragem de 1.8 ms da plataforma inercial.

A soluação escolhida para o problema foi, utilizando a técnica de decimação, aumentar a taxa de amostragem por meio de um pós-processamento dos dados, no MATLAB. Como a quantidade

Figura 4.6: Tela do simulador quando o botão GPS é acionado.

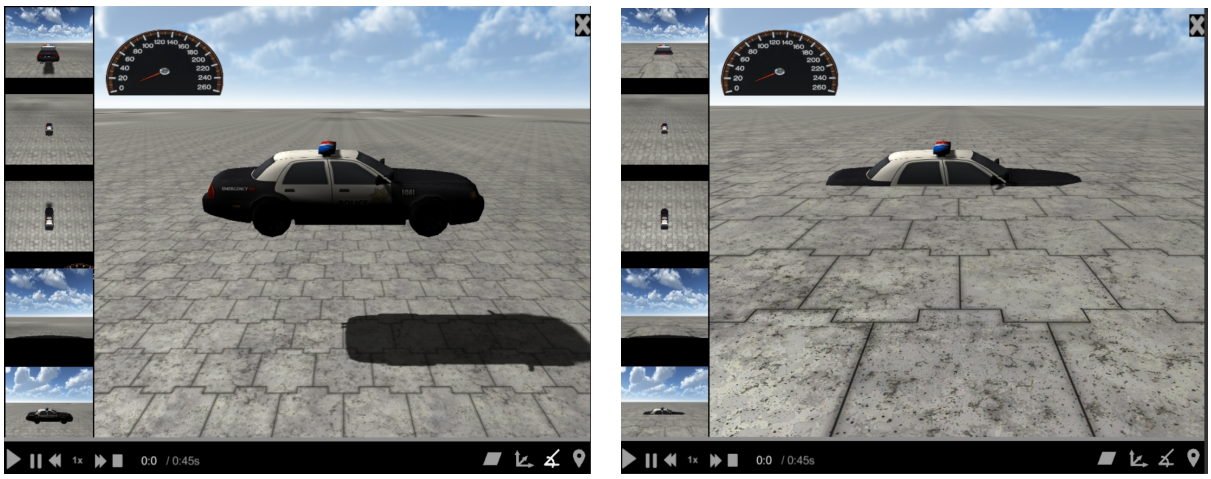
de dados sempre era um múltiplo de 384, foi escolhido fazer a decimação multiplicando a taxa de amostragem de 1.8 ms por divisores de 384. Dessa forma, a menor taxa que pode ser utilizada e que o simulador é capaz de reproduzir é de aproximadamente 28,8 ms.

Como a configuração *targetFrameRate*, específica na linguagem utilizada no Unity, só aceita argumentos inteiros, foi adotada a taxa de 35 fps (28,57 ms) para a representação em tempo real (velocidade 1x).

4.6.2 Dificuldades e Desafios Econtrados

Devido à dificuldade de implementação, o plano construído para dar sustentação ao carro não se modifica ao longo da simulação. Dessa forma, movimentos de translação no eixo Z configuram uma representação distorcida, dado que o carro passará a flutuar, no caso de movimentações no sentido positivo do eixo, ou afundar no chão, no caso de movimentações no sentido negativo do eixo, como pode ser verificado na Fig. 4.7. Esse mesmo problema também é observado no caso de movimentos de rotação nos eixos Y e Z, Fig. 4.8.

Para que essas limitações não atrapalhassem a análise do acidente, foi implementado o botão descrito na seção 4.4.1, em que é possível habilitar ou desabilitar a visualização do plano de sustentação.



(a) Positivo (b) Negativo

Figura 4.7: Problemas relativos à movimentos de translação realizados no eixo Z.

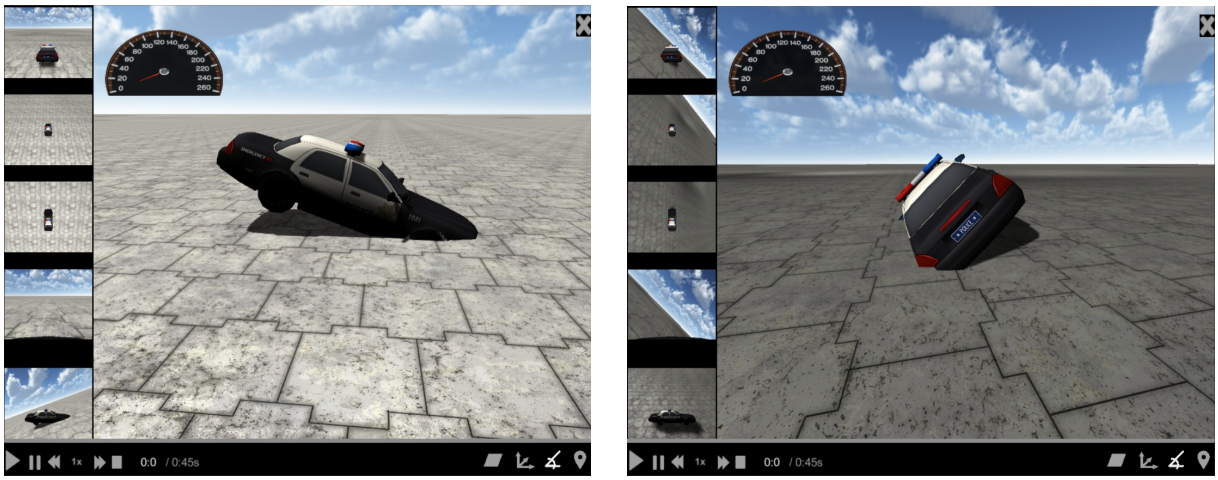
4.6.3 Velocidade de simulação

Como dito anteriormente, as velocidades disponíveis para visualização da simulação são: 0.25x, 0.5x, 1x, 2x e 4x. Para os casos em que a velocidade foi reduzida, a taxa de renderização foi diminuída. Porém, como dito anteriormente, a configuração a *targetFrameRate* só aceita valores inteiros, o que implicou na utilização de 18 FPS para representar 0,5x e 9 FPS para 0,25x.

No caso em que a velocidade deveria aumentar, foi necessário diminuir a quantidade de dados apresentados na simulação. Dessa forma, para a representação em 2x, é utilizada a taxa de 35 fps, porém somente os índices múltiplos de 2 são lidos. O mesmo é válido para a velocidade 4x, com a diferença de que os índices lidos são os múltiplos de 4. Esse método é aplicado a todos os dados, com exceção das medidas de posição translacional e angular. Isso porque os valores das posições são interpretados de forma incremental - ou seja, a não leitura de uma medida acarreta em posições finais errôneas. Dessa forma, quando a simulação acontece nas velocidades 2x e 4x, as medidas de posição que seriam "puladas"são somadas às que são lidas.

4.6.4 Tempo total de simulação

Como a taxa de renderização oscila ao longo da simulação, não é possível prever qual será o tempo total exato de execução. Dessa forma, o valor apresentado no menu inferior da tela é medido multiplicando a quantidade de dados que serão lidos pelo inverso da taxa de renderização definida na configuração *targetFrameRate* para a velocidade 1x - nesse caso, 35 fps.



(a) Eixo X

(b) Eixo Y

Figura 4.8: Problemas relativos à movimentos de rotação nos eixos X e Y.

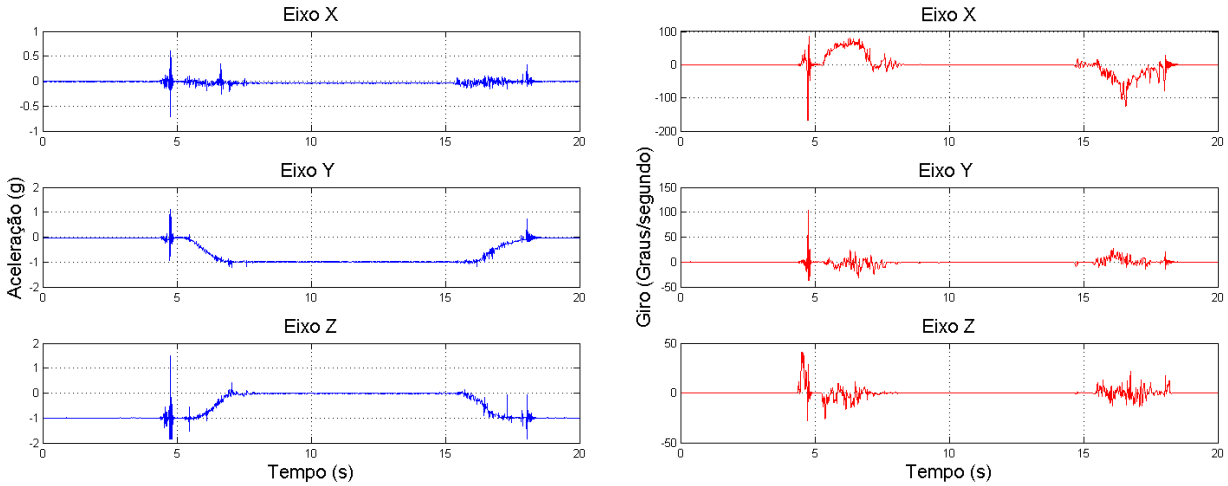
Capítulo 5

Análise dos dados obtidos pela plataforma inercial

Neste capítulo procura-se aplicar diferentes filtros sobre os dados obtidos pelo MPU e GPS, analisar graficamente as posições e velocidades da plataforma inercial obtidas e tentar minimizar o efeito da aceleração da gravidade.

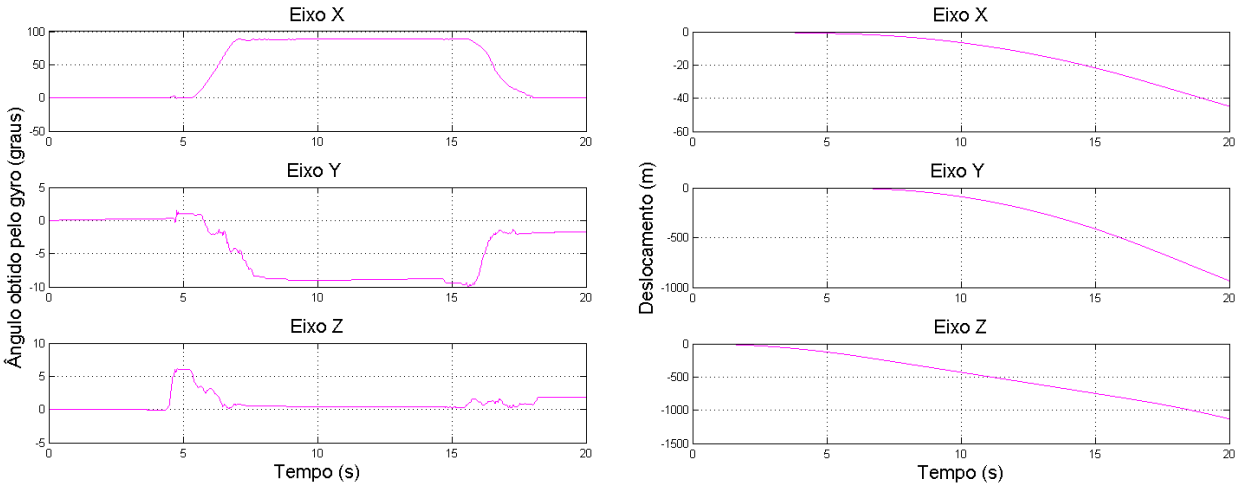
Para os dados de aceleração e giro obtidos pelo o MPU terem sentido físico coerente com a realidade, eles precisam passar por uma etapa de pós-processamento, que envolve filtragem, adição dos *offsets* para aceleração e giro obtidos, e minimização do efeito da aceleração da gravidade. Para ilustrar essa necessidade, considere os gráficos da Figuras 5.1a e 5.1b, com dados obtidos rotacionando-se a plataforma inercial 90° ao redor do eixo X, evitando fazer qualquer movimento de translação.

É notável que existe uma alta quantidade de ruído nos dados, por isso, a aplicação de um filtro torna-se essencial. Outro ponto importante, é que inicialmente a aceleração da gravidade era de aproximadamente -1 g no eixo X e após a rotação, passou a ser -1 g no eixo Y. A implicação indesejada desse efeito fica clara o gráfico da Fig. 5.2b, na qual o deslocamento nos eixos X e Y foram na ordem de 1 km após 20s. Tal deslocamento foi obtido integrando-se as acelerações numericamente, nos três eixos. O gráfico da Fig. 5.2a também mostra os ângulos de rotação da plataforma, obtidos integrando-se numericamente o giro, obtido pelo MPU. Como consequência da análise dos gráficos das Figuras 2.9 e 5.2, surge-se a importância de minimizar os efeitos da aceleração gravitacional e aplicar filtros sobre os dados obtidos.



(a) Aceleração (b) Giro

Figura 5.1: Aceleração e giro obtidos ao se rotacionar a placa 90 graus ao redor do eixo X



(a) Ângulo (b) Posição

Figura 5.2: Aceleração e giro obtidos ao se girar a placa 90 graus ao redor do eixo X

5.0.1 Aplicação de filtros

O propósito de um filtro é remover sinais indesejados do sinal que estamos medindo, que nesse caso, são as acelerações e giros obtidos pelo MPU. No caso dos dados obtidos pelo GPS, filtragem não torna-se necessária pois o algorítimo embarcado no próprio módulo já faz esse papel.

O primeiro filtro a ser analisado será o média móvel, na qual é possível ter como variável o número k de amostras tomadas recentemente, como mostra a Eq. 2.6. Ainda considerando a mesma situação mostrada nas Figuras 2.9 e 5.2, a Fig. 5.3 apresenta diferentes valores de k, aplicado a aceleração no eixo Z da Fig. 5.1a.

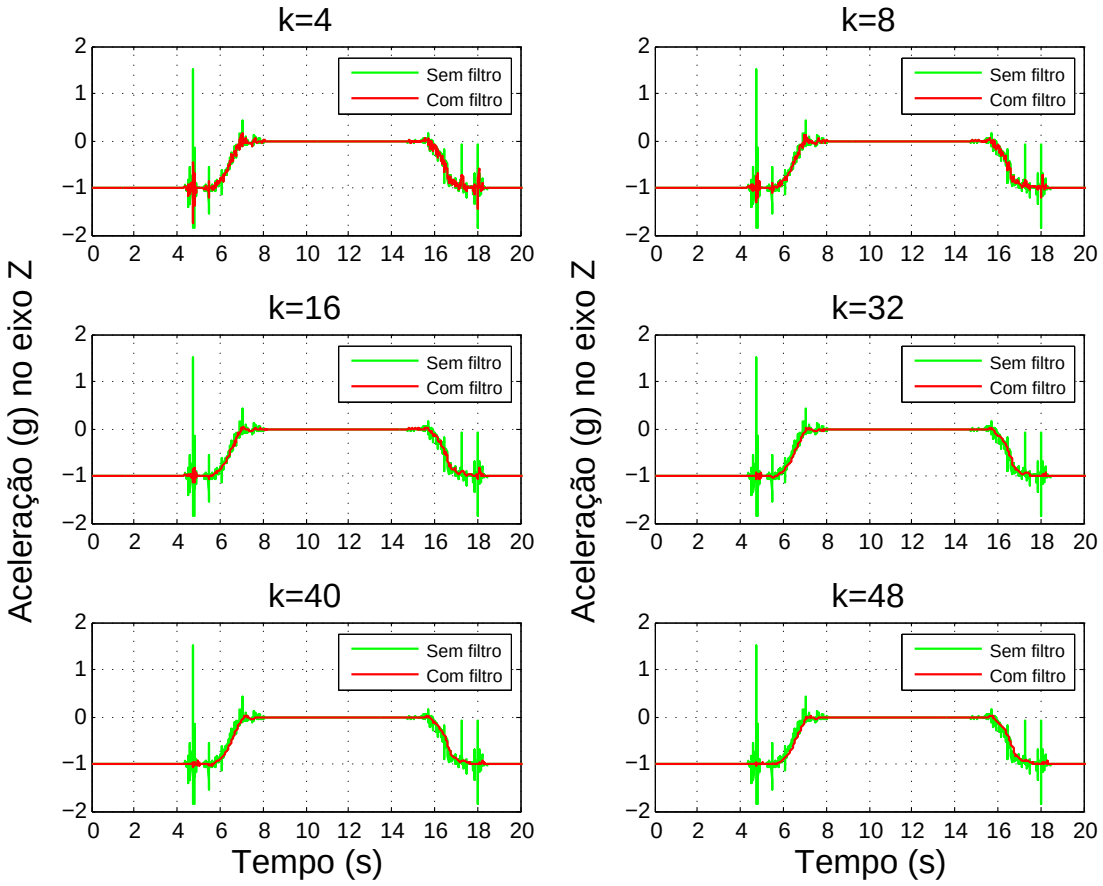


Figura 5.3: Filtro média móvel para diferentes valores de k

É notável que a aplicação do filtro média móvel, para k>32, diminuiu-se significativamente o ruido. Porém, o sinal começa a apresentar um atraso, característica comum de um filtro média móvel para altos valores de k.

Uma segunda alternativa de filtro que pode ser aplicado é o Filtro de Kalman, que como mostrado anteriormente, é um filtro probabilístico. A maior dificuldade ao se usar filtro de Kalman é estimar o matriz de covariância do erro da estimação ($\mathbf{P}$), matriz de covariância de ruído do processo

(**Q**) e a matriz de covariância de medida do processo (**R**). Para facilitar os cálculos, vamos aplicar o filtro de Kalman para acada variável, tornando as matrizes da Secção 2.5.3 unidimensionais, é obtendo-se então:

$$x_k = x_{k-1} \tag{5.1}$$

$$P_{k+1} = P_k + Q \tag{5.2}$$

$$\hat{x}_{k+1} = x_k + k \cdot (x_k - \hat{x}_i) \tag{5.3}$$

$$P_{k+1} = (1 - K) \cdot P_k \tag{5.4}$$

$$\tag{5.5}$$

Estimar com precisão a covariância do erro da estimação não é tão importante, pois, o próprio filtro de Kalman consegue minimizar os efeitos da estimação. De acordo com [33], pode ser adotado P=1 para o acelerômetro. A matriz de covariância de ruído do processo é dada pelo desvio padrão ao quadrado, que no caso das acelerações do MPU, são em média 0.001 g. A Fig. 5.4 mostra que quanto maior o valor covariância de medida do processo, melhor a eliminação dos ruídos, porém, mais atrasado o sinal fica.

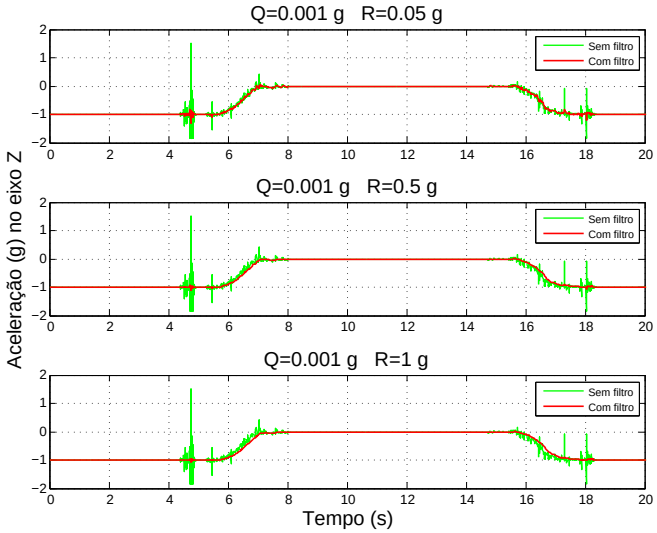


Figura 5.4: Comparação entre os filtros média móvel, Kalman e Gaussiano

Outro filtro que pode ser utilizado é o filtro Gaussiano, mostrado na Secção 2.5.4. Olhando-se o manual do MPU6050 foi possível ter uma noção da frequência de corte para um filtro passa baixa Gaussiano a ser usado nos dados obtidos pelo acelerômetro e giroscópio. Desse modo, foi possível comparar os três filtros, como mostrado na Fig. 5.5. O filtro Gaussiano foi o que apresentou o melhor resultado, pois eliminou o ruído e não ficou atrasado em relação ao sinal original. Por isso, esse foi o filtro escolhido para continuar a análise dos resultados.

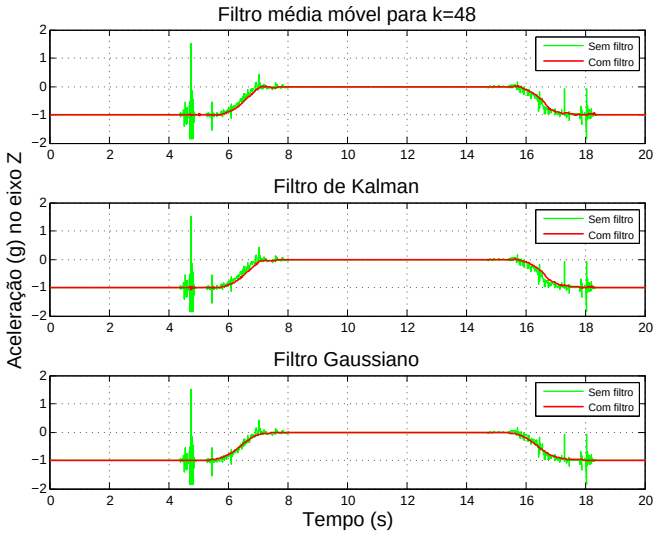


Figura 5.5: Comparação entre os filtros média móvel, Kalman e Gaussiano

5.0.2 Minimizando os efeitos da aceleração da gravidade

Como mostrado no Capítulo 2, um dos problemas ao se usar dados do acelerômetro para obter velocidade e posição, é o indesejado efeito da aceleração da gravidade. Uma forma de minimizar a influência do mesmo, é usar os ângulos de Euler e conhecendo-se alguma posição que a plataforma inercial está em repouso em relação a Terra, rotacionar o vetor aceleração da gravidade a cada ciclo de amostragem, para a nova posição angular. Porém, como foi visto na Secção 2.6, para diferentes sequências de rotações, configurações indesejadas do objeto a ser rotacionado podem ser encontradas. A solução, é trabalhar com ângulos pequenos, tendendo a zero. Considere por exemplo, a sequência de rotação R_{xyz}, dada pela Eq. 2.19, para $\theta = \phi = \psi = 0.1°$ e precisão de quatro casas decimais:

$$\mathbf{R_{xyz}} = \begin{bmatrix} \cos 0.1 \cos 0.1 & \cos 0.1 \sin 0.1 & -\sin 0.1 \\ \cos 0.1 \sin 0.1 \sin 0.1 - \cos 0.1 \sin 0.1 & \cos 0.1 \cos 0.1 + \sin 0.1 \sin 0.1 \sin 0.1 & \cos 0.1 \sin 0.1 \\ \cos 0.1 \cos 0.1 \sin 0.1 + \sin 0.1 \sin 0.1 & \cos 0.1 \sin 0.1 \sin 0.1 - \cos 0.1 \sin 0.1 & \cos 0.1 \cos 0.1 \end{bmatrix}$$

$$\mathbf{R_{xyz}} = \begin{bmatrix} 1 & 0.0017 & -0.0017 \\ -0.0017 & 1 & 0.0017 \\ 0.0017 & -0.0017 & 1 \end{bmatrix}, \tag{5.6}$$

e agora para a outra sequência R_{yxz}, dada pela Eq. 2.21, temos

$$\mathbf{R_{yxz}} = \begin{bmatrix} \cos 0.1 \cos 0.1 & \cos 0.1 \sin 0.1 & -\sin 0.1 \\ \cos 0.1 \sin 0.1 \sin 0.1 - \cos 0.1 \sin 0.1 & \cos 0.1 \cos 0.1 + \sin 0.1 \sin 0.1 \sin 0.1 & \cos 0.1 \sin 0.1 \\ \cos 0.1 \cos 0.1 \sin 0.1 + \sin 0.1 \sin 0.1 & \cos 0.1 \sin 0.1 \sin 0.1 - \cos 0.1 \sin 0.1 & \cos 0.1 \cos 0.1 \end{bmatrix}$$

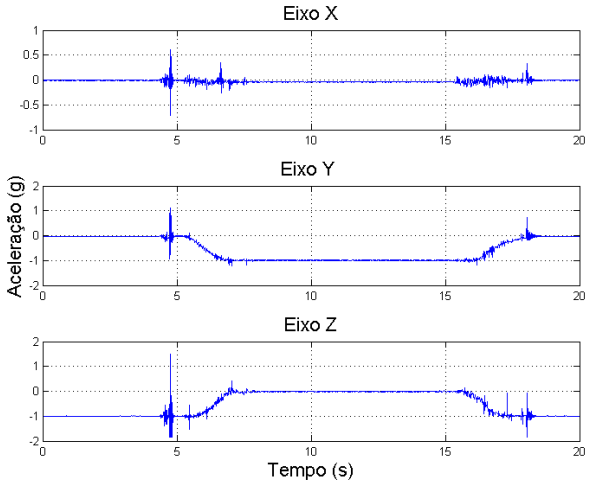
$$\mathbf{R_{yxz}} = \begin{bmatrix} 1 & 0.0017 & -0.0017 \\ -0.0017 & 1 & 0.0017 \\ 0.0017 & -0.0017 & 1 \end{bmatrix}, \tag{5.7}$$

então desse modo, $\mathbf{R_{xyz}} = \mathbf{R_{yxz}}$. Graças à esse resultado, é possível criar uma rotina que minimize os efeitos da aceleração da gravidade, como mostrado à seguir:

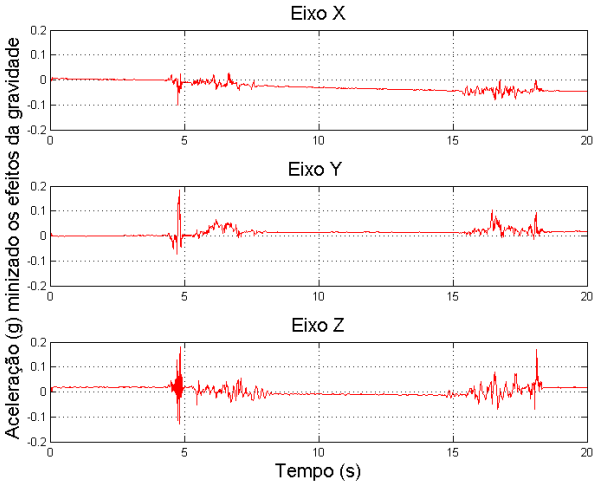
```
Gold = [0 0 -1]'; %Considera o vetor aceleracao da gravidade esta exatamente nesse
    sentido. As unidades da aceleracao estao em g.
A=[0 0 0]'; % Novo vetor aceleracao

for i = 0:(maxRows-1) % maxRows = Numero de dados da aceleracao

   roll =  gx(1,i)*(pi/180)*samplingRate;
   pitch=  gy(1,i)*(pi/180)*samplingRate;
   yaw =  gz(1,i)*(pi/180)*samplingRate; % samplingRate = periodo de amostragem

   R = angle2dcm(yaw, pitch, roll,'zyx'); %Cria a matriz rotacional
   Gnew=R*Gold; % Gnew e o novo vetor aceleracao da gravidade
   Gold=Gnew;

   A=([ax(i) ay(i) az(i)]')- Gnew;

end
```

Para ilustrar a aplicação do código acima, considere o mesmo exemplo da secção anterior, onde rotacionou-se a plataforma inercial 90° em torno do eixo x. A Fig. 5.6b ilustra o efeito de compensação de aceleração da gravidade. Além disso, integrando-se numericamente essas acelerações é possível obter os deslocamentos da 5.7.

É possível observar que, o deslocamento no eixo Y, por exemplo, era de aproximadamente -1000 km e passou para 20 m, o seja, uma melhora significativa, considerando que o correto seria aproximadamente 0 m. As principais desvantagens dessa estratégia são a necessidade de se saber a posição inicial da plataforma inercial e também, a quantidade de ruído que é adicionado nas regiões de mudanças bruscas dos valores da aceleração da gravidade. No caso do ruído, como mostrado anteriormente, a situação pode ser contornada pela filtragem dos sinais, que nesse caso não foi utilizada para verificar a eficácia da técnica em si.

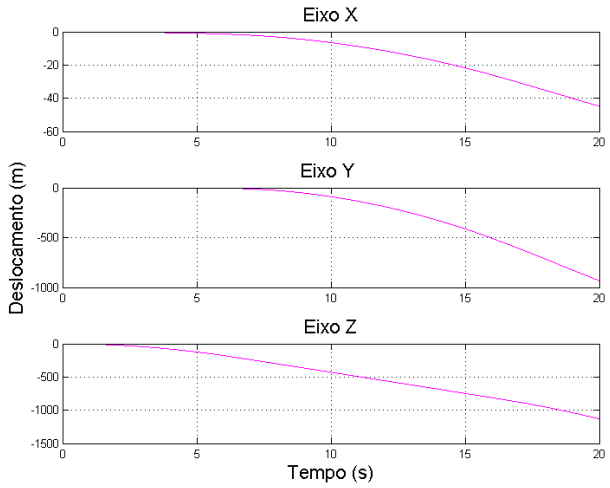


(a) Acelerações sem compensação da aceleração da gravidade

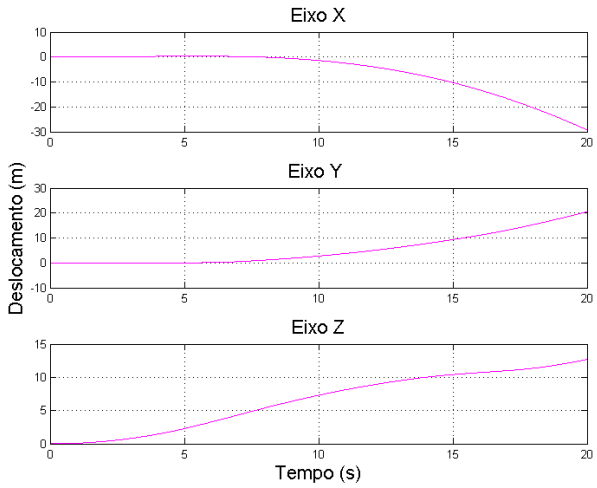


(b) Acelerações com compensação da aceleração da gravidade

Figura 5.6: Acelerações minimizando o efeito da aceleração da gravidade



(a) Posições sem compensação da aceleração da gravidade



(b) Posições com compensação da aceleração da gravidade

Figura 5.7: Posição da plataforma com e sem compensação da aceleração da gravidade

5.0.3 Testes em campo

Com o objetivo de testar o funcionamento da plataforma inercial instalada em um veículo, escolheu-se fazer um experimento no estacionamento do estádio Mané Garrincha, que é um local com bastante espaço e com pouca movimentação de pessoas, o que minimiza o risco de acidentes. O experimento consistiu em acelerar o carro do repouso até 40 km/h e em seguida, freiar bruscamente o veículo.

Figura 5.8: Plataforma inercial instalada em um veículo.

Para tal teste, uma câmera foi instalada no vidro direito ao motorista, de forma a filmar a trajetória do veículo (ver Fig. 5.9). A ideia inicial era que essa câmera filmasse também o velocímetro do carro, porém, devido ao contraste entre as iluminações dentro e fora do veículo, foi necessário outra câmera que somente filmasse o velocímetro. Tal câmera foi segurada pelo passageiro do veículo, que também tinha que checar se existia sinal no GPS, vendo se o LED azul piscava e também apertar o botão na plataforma inercial que força uma batida. Após o botão ser apertado, foram obtidos 18,66 s de dados do MPU e GPS.

Figura 5.9: Câmera instalada para filagem da trajetória do veículo.

Para esse teste o maior desafio foi conseguir de fato obter sinal do GPS e fazer o experimento dentro da janela de 18,66 s. Após cada teste era necessário conectar o Arduino no computador

via cabo USB, passar dos dados da memória EEPROM para o computador e verificar se de fato houve sinal de GPS. O terreno do estacionamento era visivelmente não plano e foi constatado no gráfico da Fig. 5.10, que mostra a inclinação da plataforma inercial, que seria aproximadamente equivalente a inclinação do terreno. Após alguns testes, foi possível obter a trajetória da Fig. 5.11, que usa dos valores de latitude e longitude obtidos pelo GPS.

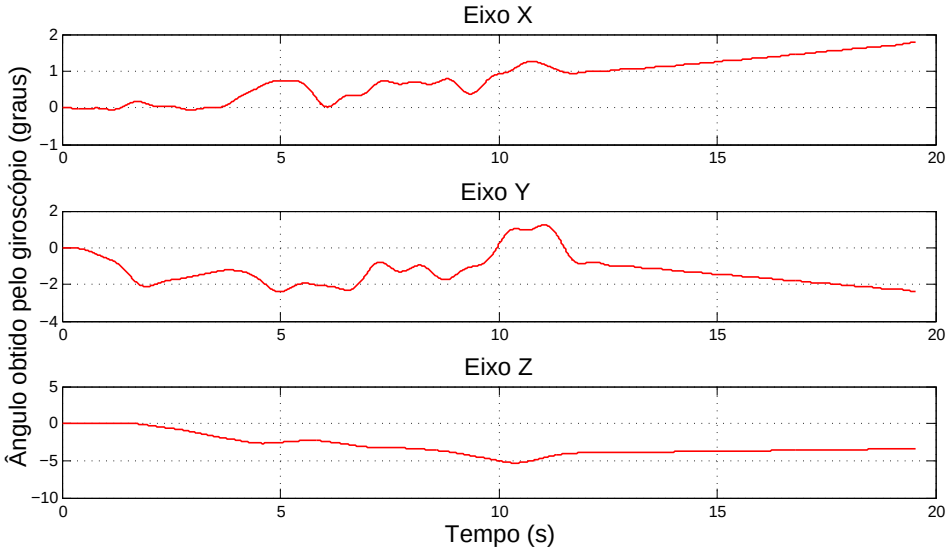


Figura 5.10: Ângulos obtidos pelos giroscópio

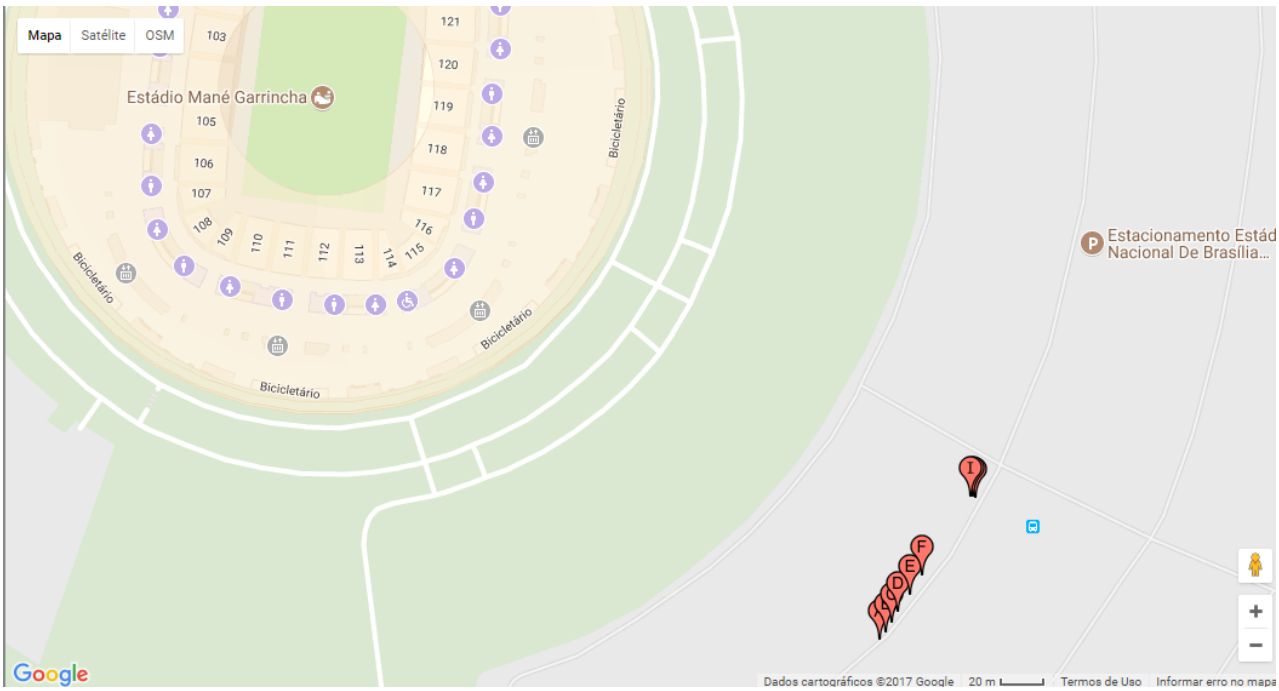


Figura 5.11: Trajetória do veículo.

O primeiro resultado importante, é que a quantidade de ruído nos dados obtidos pelo MPU foi relativamente alta, como observado nas Figuras 5.12 e 5.13, que mostram as acelerações nos três

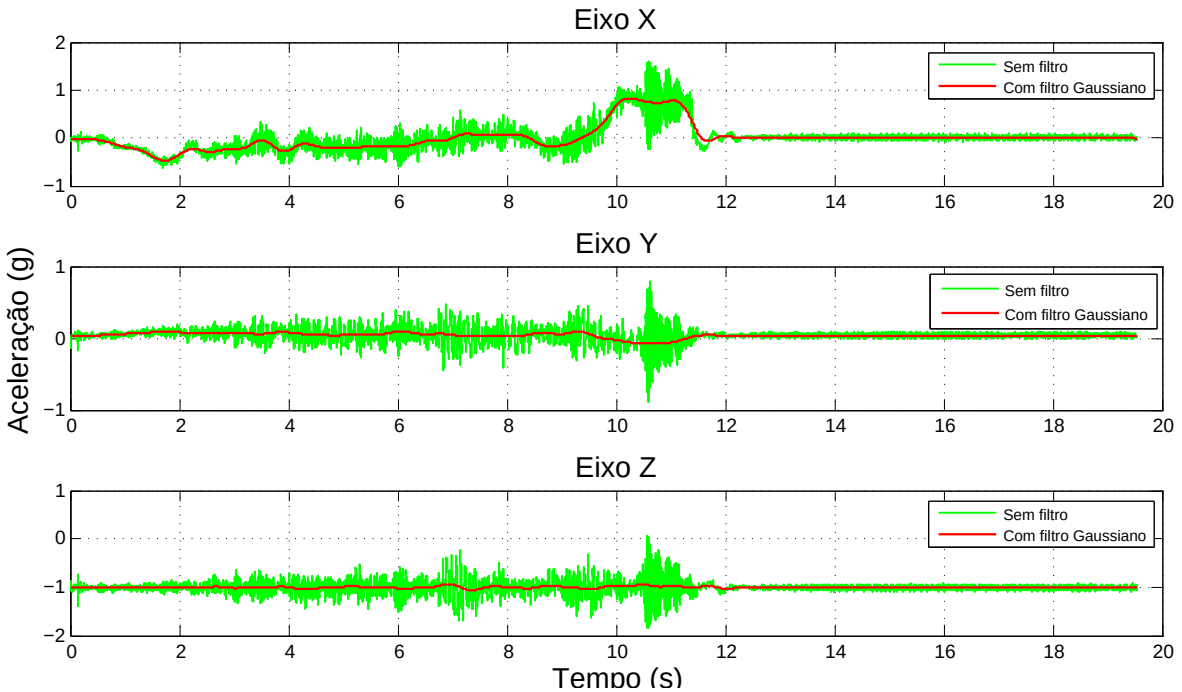


Figura 5.12: Acelerações obtidas em campo pelo MPU

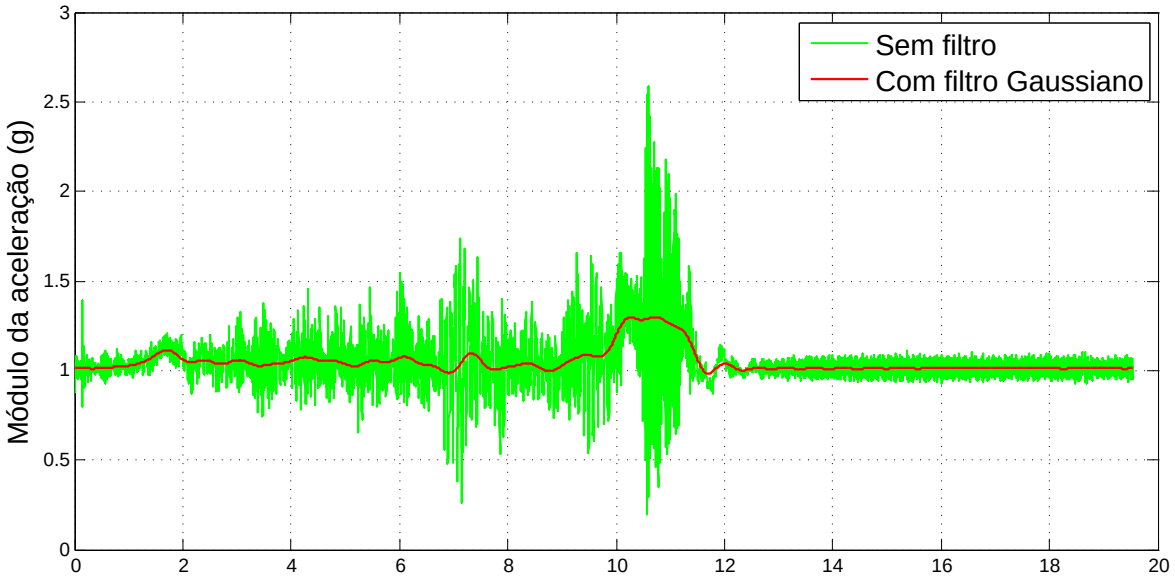


Figura 5.13: Módulo do vetor aceleração

eixos e o módulo[1] do vetor aceleração, respectivamente. No instante de tempo t= 11 s, o módulo da aceleração teve seu pico máximo, que foi causado pela frenagem do veículo. Já no instante de tempo t= 13 s, o carro estava em repouso e com o motor ligado, o que justifica a quantidade de ruído encontrado. Observe também que, nesse instante de tempo o módulo da aceleração é aproximadamente 1, que como mencionado, é causado pela aceleração da gravidade.

Para analisar se os dados do MPU estavam de acordo com os dados obtidos pelo GPS, foi feito

[1]O módulo de um vetor, de componentes (a_x,a_y,a_z), por ser definido como $\sqrt{a_x + a_y + a_z}$

o gráfico da Fig. 5.14, onde é possível notar que o traço das curvas das velocidades da plataforma inercial obtidas pelo MPU e pelo GPS são parecidos. Nos primeiros segundos, a curva em vermelho (GPS) tem um atraso em relação a curva em azul (MPU). Tal resultado é esperado, visto que, o GPS possuí um atraso devido ao seu processamento de dados e por seu período de amostragem ser alto, de em média 1 s. Com o passar do tempo, devido aos erros ao se obter velocidade por integração numérica e devido a não compensação da gravidade, o MPU começa a apresentar um resultado bem diferente do GPS. Enquanto a velocidade obtida pelo GPS chega no máximo a 38.61 km/h, a obtida pelo MPU chega a 46.44 km/h. Além disso, após 13 s, a velocidade obtida pelo MPU começa a decrescer, ao invés de ficar nula, que é outra consequência da não compensação da gravidade.

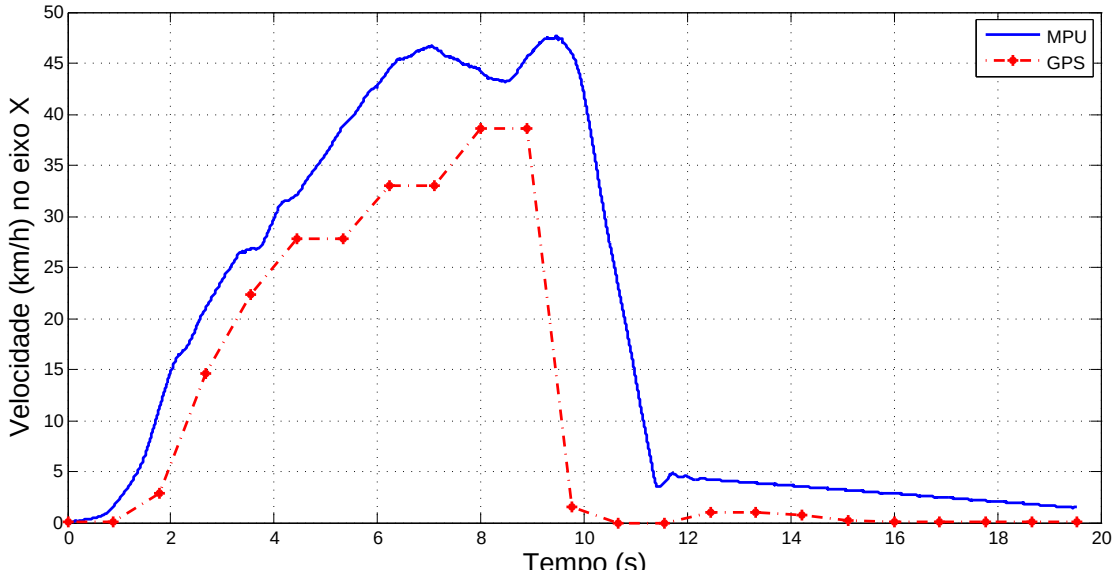


Figura 5.14: Comparação entre as velocidades da plataforma inercial obtidas pelo MPU e GPS

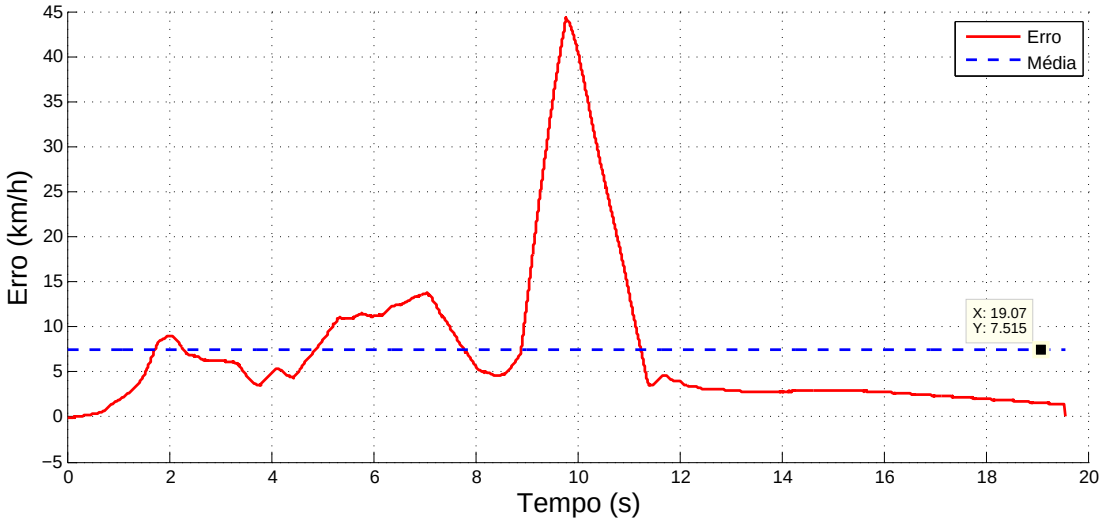


Figura 5.15: Erro entre as velocidades obtidas pelo MPU e GPS

A Fig. 5.15 mostra o erro entre as velocidades obtidas pelo MPU e pelo GPS. O erro médio foi

menor que 8 km/h e seu pico máximo foi no momento da frenagem. Tentou-se usar a técnica de compensação da aceleração da gravidade exemplificada na Secção 5.0.2, porém, como nesse caso as variações de aceleração e giro são consideráveis, os resultados obtidos ficaram muito incompatíveis com a realidade.

Capítulo 6

Conclusões

Diante a dificuldade atual de se obter vestígios que permitam ao perito criminal investigar as causas de acidentes de trânsito, foi proposta uma plataforma inercial, de baixo custo e que possa ser utilizada em diferentes veículos automotores para armazenar informações, como aceleração e velocidade, antes e depois de acidentes de trânsito.

A sistema proposto anteriormente [8] era formado por um Arduino Mega, um MPU-6050 e uma memória EEPROM, e agora foi acrescentado a uma memória SRAM e um módulo GPS. A memória SRAM permite que sistema proposto tenha uma vida útil mais elevada. Já o GPS, para quando existe sinal disponível, oferece geo-posicionamento com uma acurácia de aproximadamente 2,5 m, e mais especificamente, fornece informações de longitude, latitude e altitude, além de disponibilizar o horário da batida ao usuário. A adição do GPS também é de grande importância para fornecer a velocidade da plataforma inercial, permitindo que seus valores sejam comparados aos dados obtidos pelo MPU.

Em relação aos resultados obtidos em campo, percebeu-se uma grande quantidade de ruído nos dados obtidos pelo MPU, principalmente nas acelerações. Por isso, foi analisada a utilização de três filtros diferentes: média móvel, filtro de Kalman e filtro Gaussiano. Todos apresentaram resultados satisfatórios, porém, escolheu-se a utilização do último, por eliminar de forma eficaz o ruido dos dados de aceleração e giro obtidos pelo MPU.

Os testes em campo também mostraram que as velocidades obtidas pelo MPU e pelo GPS tiveram um erro médio de 8 km/h. O ideal seria obter a velocidade pelo MPU com boa precisão, pois nem sempre existe sinal GPS disponível. Todavia, para alcançar tal objetivo é necessário minimizar os efeitos negativos causados pela aceleração da gravidade. Tentou-se utilizar rotações sucessivas do vetor aceleração da gravidade por meio dos ângulos de Euler, porém a técnica se mostrou bem limitada e que ainda precisa ser melhorada.

6.1 Perspectivas Futuras

Para trabalhos futuros se propõe expandir a capacidade das memórias EEPROM e SRAM, de modo que facilite a fase de testes de dispositivo. O sistema atual consegue armazenar somente uma janela de 18,66 s de dados do MPU (aceleração e giro) e do GPS (horário local, latitude, longitude, altitude, etc), o que limitava bastante as possibilidades de testes em campo.

Também seria interessante a adição de um magnetômetro, que praticamente não alteraria o preço da proposa de plataforma, e que seria de grande utilidade, pois, o vetor campo magnético e o vetor aceleração da gravidade são praticamente perpendiculares e tirando seu produto vetorial, proporcionariam um sistema vetorial fixo, que é utilizado na maioria dos IMU's atuais.

Considera-se como uma das principais melhorias necessária para o sistema, aperfeiçoar o algoritmo de pós-processamento, de modo a minimizar os efeitos indesejáveis causados pela aceleração da gravidade. Para isso, seria interessante usar rotações com quatérnios, que são expansões dos números complexos para três dimensões, ao invés dos ângulos de Euler, para se evitar as perdas de graus de liberdade (*Gimbal lock*) e rotações não comutativas. Existem diversos trabalhos na literatura [34, 35, 36, 37] que demonstram a compensação da aceleração da gravidade, principalmente na área da robótica.

Mesmo com o estacionamento do estágio Mané Garrincha não apresentando um asfalto de má qualidade, a quantidade de ruído na aceleração e no giro encontrado nos resultados foi consideravelmente alta. Por isso, sugere-se também fazer mais testes em campo, com veículos diferentes e em terrenos com solos irregulares.

Por fim, no caso do Simulador 3D, necessita-se que a trajetória percorrida pelo carro seja marcada no chão, pois isso auxiliaria perceber os motivos feitos pelo veículo. Outra proposta interessante seria o Simulador obter o ambiente do acidente de trânsito pelo *Google Street View*, o que deixaria a o Simulador consideravelmente mais realista.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] INVENSENSE. *MPU-6000 and MPU-6050 Register Map and Descriptions Revision 4.2*. [S.l.].

[2] IBGE. *Recomendações para levantamentos relativos estátcos - GPS*. [S.l.], 2008. Disponível em: `ftp://geoftp.ibge.gov.br/metodos_e_outros_documentos_de_referencia/normas/recom_gps_internet.pdf`.

[3] ORGANIZATION, W. H. *Global status report on road safety 2015*. [S.l.]: World Health Organization, 2015.

[4] CARDOSO, C. *Brasil é o quarto país com mais mortes no trânsito na América, diz OMS*. Disponível em: `http://www1.folha.uol.com.br/cotidiano/2016/05/1772858-brasil-e-o-quarto-pais-com-mais-mortes-no-transito-na-america-diz-oms.shtml`.

[5] MANNERING, F. L.; BHAT, C. R. Analytic methods in accident research: Methodological frontier and future directions. *Analytic methods in accident research*, Elsevier, v. 1, p. 1–22, 2014.

[6] CURITIBA, P. C. I. de Criminalística de. *Acidentes de Trânsito*. Disponível em: `http://www.ic.pr.gov.br/modules/conteudo/conteudo.php?conteudo=44`.

[7] KLEER, A. A.; THIELO, M. R.; SANTOS, A. d. C. K. d. A física utilizada na investigação de acidentes de trânsito. 1997.

[8] LIMA, V. d. O. Proposta de plataforma inercial para auxiliar na perícia de acidentes de trânsito. 2016.

[9] (CONTRAN), C. N. de T. *RESOLUÇÃO 380, DE 28 DE ABRIL DE 2011*. 2011. `http://www.denatran.gov.br/download/Resolucoes/RESOLUCAO_CONTRAN_380_10.pdf`.

[10] LAGERGREN, M. Modeling as a tool to assist in managing problems in health care. *Evidence Based Management in Health Care: The Role of Decision Support Systems, Australian Studies in Health Service Administration*, v. 11, n. 4, p. 17–36, 2002.

[11] RODRIGUES, J. R. et al. Modelagem e simulação de um microacelerômetro capacitivo. *Área de Sistemas Aeroespaciais e Mecatrônica, Instituto Tecnológico de Aeronáutica. São José dos Campos*, p. 219, 2011.

[12] RODRIGUES, J. R. et al. Modelagem e simulação de um microacelerômetro capacitivo. *São José dos Campos*, 2011.

[13] FORHAN, N. A. E.; SENNA, J. R. S. Giroscópios mems. *Relatório técnico-científico, Instituto Nacional de Pesquisas Espaciais-INPE, São José dos Campos-SP-Brasil, iNPE-16671-RPQ/848*, 2010.

[14] PANG, G.; LIU, H. Evaluation of a low-cost mems accelerometer for distance measurement. *Journal of Intelligent and Robotic Systems*, Springer, v. 30, n. 3, p. 249–265, 2001.

[15] MAHONY, R.; HAMEL, T.; PFLIMLIN, J.-M. Nonlinear complementary filters on the special orthogonal group. *IEEE Transactions on automatic control*, IEEE, v. 53, n. 5, p. 1203–1218, 2008.

[16] REVNIVYKH, S. Glonass: Status and perspectives. *Munich Satellite Navigation Summit*, 2005.

[17] MAURÍCIO, A. F. Q. *Global Navigation Satellite System passado, presente e futuro*. Tese (Doutorado), 2015.

[18] CARVALHO, E. A. de; ARAÚJO, P. C. de. Localização: coordenadas geográficas. 2008.

[19] SUBIRANA, J. S.; ZORNOZA, J. J.; HERNÁNDEZ-PAJARES, M. Gnss data processing. volume 1: Fundamentals and algorithms. *ESA Communications ESTEC, PO Box*, v. 299, p. 2200, 2013.

[20] PETLEY, B. New definition of the metre. *Nature*, v. 303, n. 5916, p. 373–376, 1983.

[21] DAWOUD, S. Gnss principles and comparison. *Potsdam University*.

[22] MICROSHIP. *1024K I2C CMOS Serial EEPROM*. [S.l.].

[23] OLIVEIRA, M. D. 3d visual simulation platform for the project of a new hospital facility. *Monitoring, Evaluating, Planning Health Services*, ISBN 981-02-4154-2, p. 39–52, 1999.

[24] SMITH, S. W. et al. The scientist and engineer's guide to digital signal processing. California Technical Pub. San Diego, 1997.

[25] LAGES, W. F. Filtro de kalman. *Apostila Curso de Controle Digital*, v. 7, 2005.

[26] BISHOP, G.; WELCH, G. An introduction to the kalman filter. *Proc of SIGGRAPH, Course*, v. 8, n. 27599-23175, p. 41, 2001.

[27] YOUNG, I. T.; VLIET, L. J. V. Recursive implementation of the gaussian filter. *Signal processing*, Elsevier, v. 44, n. 2, p. 139–151, 1995.

[28] PIO, R. Euler angle transformations. *IEEE Transactions on Automatic Control*, IEEE, v. 11, n. 4, p. 707–715, 1966.

[29] PEDLEY, M. Tilt sensing using a three-axis accelerometer. *Freescale semiconductor application note*, p. 1–22, 2013.

[30] 3.0, C. C. A. S. *ARDUINO MEGA 2560 REV3*. [S.l.].

[31] UBLOX. *MNEO-6 u-blox 6 GPS Modules Data Sheet*. [S.l.].

[32] MICROSHIP. *1Mbit SPI Serial SRAM with SDI and SQI Interface*. [S.l.].

[33] RODRÍGUEZ, D. V. et al. Sistemas para determinar la posición y orientación de herramientas quirúrgicas en operaciones de cirugía laparoscópica. 2014.

[34] SHOEMAKE, K. Animating rotation with quaternion curves. In: ACM. *ACM SIGGRAPH computer graphics*. [S.l.], 1985. v. 19, n. 3, p. 245–254.

[35] NISTLER, J. R.; SELEKWA, M. F. Gravity compensation in accelerometer measurements for robot navigation on inclined surfaces. *Procedia Computer Science*, Elsevier, v. 6, p. 413–418, 2011.

[36] JEKELI, C. Precision free-inertial navigation with gravity compensation by an onboard gradiometer. *Journal of Guidance Control and Dynamics*, New York: The Institute, 1982-, v. 29, n. 3, p. 704–713, 2006.

[37] ULRICH, N.; KUMAR, V. Passive mechanical gravity compensation for robot manipulators. In: IEEE. *Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on*. [S.l.], 1991. p. 1536–1541.

APÊNDICES

I. PROGRAMAS UTILIZADOS

```
////////////////////////////////////////////////////
//----------------------LIBRARIES---------------------//
////////////////////////////////////////////////////

#include <Wire.h>
#include <SPI.h>
#include <SpiRam_Extended.h>
#include <TinyGPS.h>

////////////////////////////////////////////////////
//----------------------CONSTANTS---------------------//
////////////////////////////////////////////////////

//-----------------LEDs and BUTTON---------------------//
#define LED_VM  A10   // Red
#define LED_AM  A9    // Yellow
#define LED_VD  A8    // Green
#define LED_AMO 13    // Arduino's organge LED
#define BUTTON_PIN 3

//-----------------MEMORY DISTRIBUTION-----------------//
// IA = Initial Address and FA = Final Address
#define BLOCK1_MPU_IA 256    //256
#define BLOCK1_MPU_FA 129279 //129279
#define BLOCK1_GPS_IA 129280 //129280
#define BLOCK1_GPS_FA 130432 //130432

//----------------------EEPROM-----------------------//

// General use EEPROM constants
#define EEPROM_ADDRESS1 0x50    // Address of first half 24LC256 eeprom chip
#define EEPROM_ADDRESS2 0x54    // Address of second half 24LC256 eeprom chip
#define PAGE_SIZE 128    // Size of a page in bytes
#define CHUNKS_SIZE 12  // Size of the MPU6050 data for one cycle in bytes
#define PAGE_CHUNK_SIZE (PAGE_SIZE*3) // PAGE_CHUNK_SIZE = 384
#define NUMBER_OF_PAGE_CHUNKS ((BLOCK1_MPU_FA-BLOCK1_MPU_IA)/PAGE_CHUNK_SIZE) //
    Maximum is 341
#define NUMBER_OF_CHUNKS (NUMBER_OF_PAGE_CHUNKS*32) // Maximum is 5456
#define CONSTANT 0

//----------------------MPU6050----------------------//

//Offsets for MPU6050 calibration
#define AX_OFFSET_DEF -1630
#define AY_OFFSET_DEF 191
#define AZ_OFFSET_DEF 2574
#define GX_OFFSET_DEF 719
```

```
#define GY_OFFSET_DEF 170
#define GZ_OFFSET_DEF -24

// Shooting limits in g
#define SHOOT_AX_G 2.0      //Accelaration shooting limit on X axis (g unit)
#define SHOOT_AY_G 2.0
#define SHOOT_AZ_G 2.0
#define SHOOT_GX_G 5000.0
#define SHOOT_GY_G 5000.0
#define SHOOT_GZ_G 5000.0

// Shooting limits in MPU unit
#define SHOOT_AX (SHOOT_AX_G*16384)
#define SHOOT_AY (SHOOT_AY_G*16384)
#define SHOOT_AZ (SHOOT_AZ_G*16384)
#define SHOOT_GX (SHOOT_GX_G*131)
#define SHOOT_GY (SHOOT_GY_G*131)
#define SHOOT_GZ (SHOOT_GZ_G*131)

#define MPUaddress 0x68 // MPU6050 I2C address
#define SS_PIN 53       // Pin CS


//////////////////////////////////////////////////////
//-------------------STRUCTURES---------------------//
//////////////////////////////////////////////////////

union DataUnion
{
    unsigned long ulongVet[CHUNKS_SIZE/2];
    long longVet[CHUNKS_SIZE/2];
    float floatVet[CHUNKS_SIZE/2];
    int16_t int16Vet[CHUNKS_SIZE];
    int intVet[CHUNKS_SIZE];
    byte bytesVet[CHUNKS_SIZE*2];
    char charVet[CHUNKS_SIZE*2];
};

union DataUnionPageChunk
{
    char charVet[PAGE_CHUNK_SIZE];
    int16_t int16Vet[PAGE_CHUNK_SIZE/2];
    long longVet[PAGE_CHUNK_SIZE/4];
};

//////////////////////////////////////////////////////
//-----------------GLOBAL VARIABLES-----------------//
//////////////////////////////////////////////////////

SpiRAM SpiRam(SPI_CLOCK_DIV4, SS_PIN, CHIP_23LC1024);
TinyGPS gps1;
```

```
long memLineMPU=BLOCK1_MPU_IA,memLineGPS=BLOCK1_GPS_IA, crashLineMPU, crashLineGPS;
     // Line memory for SRAM and EEPROM. Line 1 means byte 1, for example
char menuOption='P',crashState='N';
bool buttonState=HIGH;

////////////////////////////////////////////////////
//----------------------FUNCTIONS---------------------//
////////////////////////////////////////////////////

//-----------Function within this file-------------//
void saveDataToSRAM(long *memLine, DataUnion *dataUnion, long BLOCK1_IA, long BLOCK
    1_FA, long blockSize);
void displayDataUnionMPU(DataUnion *dataUnion);
void displayDataUnionGPS(DataUnion *dataUnion);
bool checkCrash(DataUnion dataUnion);
void moveDataFromSRAMtoEEPROM();
void displayEntireSRAMandEEPROM();
void displayEntireEEPROM();
void displayEntireSRAM();
void displayConfigPage();

//---------------------MPU functions-------------------//
void calibrateMPU6050();
void getMPU6050motion(DataUnion *dataUnion);

//---------------------GPS functions-------------------//
bool getGPSmotion(DataUnion *dataUnion);

//---------------------EEPROM functions----------------//
void writePageEEPROM(long address, char* data);
void readEEPROM(long address, unsigned char* data, unsigned int num_chars);
int readIntEEPROM(long address);
char readCharEEPROM(long address);
byte readByteEEPROM(long address);
long readLongEEPROM(long address);
void initizalingConfigPage();

//---------------------SRAM functions-------------------//

// See the 'SpiRam_Extended.h' file where this program was installed

void setup() {

  pinMode(LED_VD,OUTPUT);
  digitalWrite(LED_VD,LOW);
  pinMode(LED_AM,OUTPUT);
  digitalWrite(LED_AM,LOW);
  pinMode(LED_VM,OUTPUT);
  digitalWrite(LED_VM,LOW);
  pinMode(LED_AMO,OUTPUT);
  digitalWrite(LED_AMO,LOW);
```

```
148    digitalWrite(LED_VD,HIGH);
       delay(500);
150    digitalWrite(LED_VD,LOW);

152    digitalWrite(LED_AM,HIGH);
       delay(500);
154    digitalWrite(LED_AM,LOW);

156    digitalWrite(LED_VM,HIGH);
       delay(500);
158    digitalWrite(LED_VM,LOW);

160    digitalWrite(LED_AMO,HIGH);
       delay(500);
162    digitalWrite(LED_AMO,LOW);

164    pinMode(12,OUTPUT);
       digitalWrite(12,LOW);
166
       pinMode(BUTTON_PIN,INPUT_PULLUP);
168
         Wire.begin();
170      Wire.beginTransmission(MPUaddress);
         Wire.write(0x6B); // PWR_MGMT_1 register
172
         Wire.write(0); // Initializes the MPU
174      Wire.endTransmission(true);

176      Serial.begin(115200);
         SPI.begin(); // Starting SPI protocol for SRAM use
178      Serial2.begin(9600); // Serial communication for GPS use
         Serial.println("Programa inicializado");
180
         for (long j=BLOCK1_GPS_IA;j<BLOCK1_GPS_FA;j++)
182          SpiRam.write_byte(j,0);
       SpiRam.write_byte((char) 33,'N');
184    SpiRam.write_byte((char) 34,'N');
       SpiRam.write_byte((char) 35,'N');
186    SpiRam.write_byte((char) 36,'N');
       SpiRam.write_byte((char) 37,'N');
188    SpiRam.write_byte((char) 38,'N');

190 }

192 void loop() {
       bool GPStag=false;     // Tag to indicate if there's GPS data available
194    DataUnion dataUnion;   // Union of 2*CHUNKS_SIZE bytes
       buttonState=HIGH;
196
       buttonState = digitalRead(BUTTON_PIN);
198    //if(buttonState==LOW)
       // digitalWrite(LED_VM,LOW);
```

```
200     //Serial.print("buttonState = ");Serial.println(buttonState);

202     if (Serial.available() > 0){
          menuOption=Serial.read();
204       if(menuOption=='0')
            displayConfigPage();
206       if((menuOption=='C')||(menuOption=='c'))
            calibrateMPU6050();
208       if((menuOption=='I')||(menuOption=='i'))
             if(crashState='S')
210            crashState='N';
             else
212            crashState='S';
          if((menuOption=='E')||(menuOption=='e'))
214          displayEntireEEPROM();
           if((menuOption=='S')||(menuOption=='s'))
216          displayEntireSRAM();
        }
218
      getMPU6050motion(&dataUnion);
220   saveDataToSRAM(&memLineMPU,&dataUnion,BLOCK1_MPU_IA,BLOCK1_MPU_FA, CHUNKS_SIZE)
      ;
      if((menuOption=='R')||(menuOption=='r'))
222     displayDataUnionMPU(&dataUnion);

224   GPStag=getGPSmotion(&dataUnion);
      if(GPStag==true)
226     {
        saveDataToSRAM(&memLineGPS,&dataUnion,BLOCK1_GPS_IA,BLOCK1_GPS_FA,
    CHUNKS_SIZE*2);
228     if((menuOption=='R')||(menuOption=='r'))
          displayDataUnionGPS(&dataUnion);
230     digitalWrite(LED_VD,LOW);
        }
232
      if((checkCrash(dataUnion)==1)||(buttonState==LOW))
234     {
            // unsigned long cT = micros(); // count loop time
236         Serial.println("--------------Ocorreu uma batida!--------------");
            digitalWrite(LED_VM,HIGH);    // RED LED to indicate car crash
238         Serial.print("buttonState = ");Serial.println(buttonState);
            crashLineMPU=memLineMPU;
240         crashLineGPS=memLineGPS;
            crashState='S';
242
            // Creating a memory line of zeros to indicate crash
244         for(long j=0;j<(CHUNKS_SIZE*2);j++)
              dataUnion.charVet[j]=0;
246         saveDataToSRAM(&memLineMPU,&dataUnion,BLOCK1_MPU_IA,BLOCK1_MPU_FA,
    CHUNKS_SIZE);
            saveDataToSRAM(&memLineGPS,&dataUnion,BLOCK1_GPS_IA,BLOCK1_GPS_FA,
    CHUNKS_SIZE*2);
```

```
248
                if((menuOption=='R')||(menuOption=='r'))
                  displayDataUnionMPU(&dataUnion);

252             long iLimit;
                if(buttonState==LOW)
254               iLimit=NUMBER_OF_CHUNKS;
                else
256               iLimit=NUMBER_OF_CHUNKS/2;

258             for(long i=0;i<iLimit;i++) // Get the next NUMBER_OF_CHUNKS/2 or
    NUMBER_OF_CHUNKS number of data
                {
260                 getMPU6050motion(&dataUnion);
                    saveDataToSRAM(&memLineMPU,&dataUnion,BLOCK1_MPU_IA,BLOCK1_MPU_FA,
    CHUNKS_SIZE);
262                 if((menuOption=='R')||(menuOption=='r'))
                      displayDataUnionMPU(&dataUnion);
264
                    GPStag=getGPSmotion(&dataUnion);
266                 if(GPStag==true)
                      {
268                   if((menuOption=='R')||(menuOption=='r'))
                        displayDataUnionGPS(&dataUnion);
270                   saveDataToSRAM(&memLineGPS,&dataUnion,BLOCK1_GPS_IA,BLOCK1_GPS_FA
    ,CHUNKS_SIZE*2);}
                }// End of for loop
272
          // unsigned long pT = micros();
274       // unsigned long dT = pT - cT;
          // Serial.print("- Tempo total de coleta de dados =  ");
276       // Serial.print(dT/1000.000);Serial.println(" ms");
              digitalWrite(LED_VM,LOW);
278
           for(long j=0;j<(CHUNKS_SIZE*2);j++)
280           dataUnion.charVet[j]=0;
          saveDataToSRAM(&memLineGPS,&dataUnion,BLOCK1_GPS_IA,BLOCK1_GPS_FA,
    CHUNKS_SIZE*2);
282
          initizalingConfigPage();
284       moveDataFromSRAMtoEEPROM();      // It takes about 45 s to move 128 kbytes
    from SRAM to EEPROM
          //displayEntireSRAMandEEPROM(); // Uncomment this line if you wanna
    compare the SRAM data with the EEPROM data
          digitalWrite(LED_AMO,LOW);
          // while(true);
288        }// End of if
}
290
void saveDataToSRAM(long *memLine, DataUnion *dataUnion, long BLOCK1_IA, long BLOCK
    1_FA, long blockSize){
292
```

```
  if(*memLine>=BLOCK1_FA)
294       {*memLine=BLOCK1_IA; //Serial.println("Chegou no final do bloco de gravacao")
    ;
          }
296     SpiRam.write_stream(*memLine, dataUnion->charVet, blockSize);

298   *memLine=(*memLine)+blockSize;
}
300
void getDataFromSRAM(long memLineMPU, DataUnion *dataUnion){
302
    SpiRam.read_stream(memLineMPU, dataUnion->charVet, CHUNKS_SIZE);
304 }

306 void displayDataUnionMPU(DataUnion *dataUnion){

308 char msg[100];

310   sprintf(msg," %06d",dataUnion->int16Vet[0]); Serial.print(msg);
   sprintf(msg," %06d",dataUnion->int16Vet[1]); Serial.print(msg);
312   sprintf(msg," %06d",dataUnion->int16Vet[2]); Serial.print(msg);
   sprintf(msg," %06d",dataUnion->int16Vet[3]); Serial.print(msg);
314   sprintf(msg," %06d",dataUnion->int16Vet[4]); Serial.print(msg);
   sprintf(msg," %06d",dataUnion->int16Vet[5]); Serial.println(msg);
316 }

318 void displayDataUnionGPS(DataUnion *dataUnion){

320 char msg[100];

322   /* Serial.println("");
      Serial.println("GPS DATA UNION:");
324     Serial.print("Data (GMT): ");
      Serial.print(dataUnion->bytesVet[3]);Serial.print("/");
326     Serial.print(dataUnion->bytesVet[2]); Serial.print("/");
      Serial.println(dataUnion->intVet[0]);
328     Serial.print("Horario (GMT): ");Serial.print(dataUnion->bytesVet[4]);Serial.
    print(":");
      Serial.print(dataUnion->bytesVet[5]);Serial.print(":");
330     Serial.print(dataUnion->bytesVet[6]);Serial.print(":"); Serial.println(
    dataUnion->bytesVet[7]);
      Serial.print("Latitude: ");Serial.println(dataUnion->floatVet[2],6);
332     Serial.print("Longitude: ");Serial.println(dataUnion->floatVet[3],6);
      Serial.print("Altitude: (cm): ");Serial.println(dataUnion->floatVet[4],2);
334     Serial.print("Velocidade (km/h):");Serial.println(dataUnion->floatVet[5],2);*/

336     sprintf(msg," %04d %02d %02d %02d %02d %02d %02d ",dataUnion->intVet[0],
    dataUnion->bytesVet[2],dataUnion->bytesVet[3],dataUnion->bytesVet[4],dataUnion->
    bytesVet[5],dataUnion->bytesVet[6],dataUnion->bytesVet[7]);    Serial.print(msg);
      Serial.print(dataUnion->floatVet[2],6); Serial.print(" ");
338     Serial.print(dataUnion->floatVet[3],6); Serial.print(" ");
      Serial.print(dataUnion->floatVet[4],2); Serial.print(" ");
```

```
340       Serial.println(dataUnion->floatVet[5],2);
}
342
bool checkCrash(DataUnion dataUnion){
344
  char shoot_ax_status='N',shoot_ay_status='N',shoot_az_status='N',shoot_gx_status=
    'N',shoot_gy_status='N',shoot_gz_status='N';
346
  if(abs(dataUnion.int16Vet[0])>SHOOT_AX)
348   shoot_ax_status='S';
  if(abs(dataUnion.int16Vet[1])>SHOOT_AY)
350   shoot_ay_status='S';
//  if((abs(dataUnion.int16Vet[2])>(16384+SHOOT_AZ))||((dataUnion.int16Vet[2])
    <(16384-SHOOT_AZ)))
352 //    shoot_az_status='S';
  if(abs(dataUnion.int16Vet[2])>SHOOT_AZ)
354   shoot_az_status='S';
  if(abs(dataUnion.int16Vet[3])>SHOOT_GX)
356   shoot_gx_status='S';
  if(abs(dataUnion.int16Vet[4])>SHOOT_GY)
358   shoot_gy_status='S';
  if(abs(dataUnion.int16Vet[5])>SHOOT_GZ)
360   shoot_gz_status='S';

362 if((shoot_ax_status=='S')||(shoot_ay_status=='S')||(shoot_az_status=='S')||(
    shoot_gx_status=='S')||(shoot_gy_status=='S')||(shoot_gz_status=='S'))
  {
364   Serial.print("SHOOT_AX=");Serial.println(SHOOT_AX);
    Serial.print("SHOOT_AY=");Serial.println(SHOOT_AY);
366   Serial.print("SHOOT_AZ=");Serial.println(SHOOT_AZ);
    Serial.print("SHOOT_GX=");Serial.println(SHOOT_GX);
368   Serial.print("SHOOT_GY=");Serial.println(SHOOT_GY);
    Serial.print("SHOOT_GZ=");Serial.println(SHOOT_GZ);
370
    Serial.print("dataUnion.int16Vet[0]=");Serial.println(dataUnion.int16Vet[0]);
372   Serial.print("dataUnion.int16Vet[1]=");Serial.println(dataUnion.int16Vet[1]);
    Serial.print("dataUnion.int16Vet[2]=");Serial.println(dataUnion.int16Vet[2]);
374   Serial.print("dataUnion.int16Vet[3]=");Serial.println(dataUnion.int16Vet[3]);
    Serial.print("dataUnion.int16Vet[4]=");Serial.println(dataUnion.int16Vet[4]);
376   Serial.print("dataUnion.int16Vet[5]=");Serial.println(dataUnion.int16Vet[5]);

378   Serial.print("shoot_ax_status=");Serial.println(shoot_ax_status);
    Serial.print("shoot_ay_status=");Serial.println(shoot_ay_status);
380   Serial.print("shoot_az_status=");Serial.println(shoot_az_status);
    Serial.print("shoot_gx_status=");Serial.println(shoot_gx_status);
382   Serial.print("shoot_gy_status=");Serial.println(shoot_gy_status);
    Serial.print("shoot_gz_status=");Serial.println(shoot_gz_status);
384   SpiRam.write_byte((char) 33,shoot_ax_status);
    SpiRam.write_byte((char) 34,shoot_ay_status);
386   SpiRam.write_byte((char) 35,shoot_az_status);
    SpiRam.write_byte((char) 36,shoot_gx_status);
388   SpiRam.write_byte((char) 37,shoot_gy_status);
```

```
    SpiRam.write_byte((char) 38,shoot_gz_status);
    return(1);
  }else{
    return(0);}
}

void moveDataFromSRAMtoEEPROM(){

long iLimit;
char vet[PAGE_SIZE];


    digitalWrite(LED_VD,HIGH);
    digitalWrite(LED_AM,HIGH);
    digitalWrite(LED_VM,HIGH);

    // Configuration page
     SpiRam.read_stream((long) 0,vet, PAGE_SIZE);
     writePageEEPROM((long) 0,vet);

    // MPU data
    for (long i=BLOCK1_MPU_IA;i<BLOCK1_MPU_FA;i=i+PAGE_SIZE)
    {
      SpiRam.read_stream((long) i,vet, PAGE_SIZE);
      writePageEEPROM((long) i,vet);
    }

    // GPS data
    for (long j=BLOCK1_GPS_IA;j<BLOCK1_GPS_FA;j=j+PAGE_SIZE)
    {
      SpiRam.read_stream((long) j,vet, PAGE_SIZE);
      writePageEEPROM((long) j,vet);
    }

    digitalWrite(LED_VD,LOW);
    digitalWrite(LED_AM,LOW);
    digitalWrite(LED_VM,LOW);
}

void displayEntireSRAMandEEPROM(){

char msg[100], indicator;
DataUnion dataUnion1;
DataUnion dataUnion2;

    displayConfigPage();

    for (long i=BLOCK1_MPU_IA;i<BLOCK1_MPU_FA;i=i+CHUNKS_SIZE)
    {
          sprintf(msg,"%061d     SRAM    :",i);    Serial.print(msg);
          SpiRam.read_stream((long) i, dataUnion1.charVet, CHUNKS_SIZE);
          displayDataUnionMPU(&dataUnion1);
```

```
442           readEEPROM((long) i, dataUnion2.charVet,CHUNKS_SIZE);
              if(memcmp(dataUnion1.charVet,dataUnion2.charVet, CHUNKS_SIZE)==0)
444              {indicator='S';}
              else
446              {indicator='N';}
              sprintf(msg,"%06ld  EEPROM  ",i);Serial.print(msg);Serial.print(indicator)
    ;Serial.print(":");
448           displayDataUnionMPU(&dataUnion2);
      }
450
      for (long j=BLOCK1_GPS_IA;j<BLOCK1_GPS_FA;j=j+(CHUNKS_SIZE*2))
452   {
              sprintf(msg,"%06ld    SRAM    :",j);    Serial.print(msg);
454           SpiRam.read_stream((long) j, dataUnion1.charVet, CHUNKS_SIZE*2);
              displayDataUnionGPS(&dataUnion1);
456
              readEEPROM((long) j, dataUnion2.charVet,CHUNKS_SIZE*2);
458           if(memcmp(dataUnion1.charVet,dataUnion2.charVet, CHUNKS_SIZE*2)==0)
                 {indicator='S';}
460           else
                 {indicator='N';}
462           sprintf(msg,"%06ld  EEPROM  ",j);Serial.print(msg);Serial.print(indicator)
    ;Serial.print(":");
              displayDataUnionGPS(&dataUnion2);
464   }
}
466
void displayEntireEEPROM(){
468
DataUnion dataUnion;
470
      for (long i=BLOCK1_MPU_IA;i<BLOCK1_MPU_FA;i=i+CHUNKS_SIZE)
472   {
        readEEPROM(i, dataUnion.charVet, CHUNKS_SIZE);
474     displayDataUnionMPU(&dataUnion);
      }
476
      for (long i=BLOCK1_GPS_IA;i<BLOCK1_GPS_FA;i=i+(CHUNKS_SIZE*2))
478   {
        readEEPROM(i, dataUnion.charVet, CHUNKS_SIZE*2);
480     displayDataUnionGPS(&dataUnion);
      }
482 }

484 void displayEntireSRAM(){

486 DataUnion dataUnion;

488   for (long i=BLOCK1_MPU_IA;i<BLOCK1_MPU_FA;i=i+CHUNKS_SIZE)
      {
490     SpiRam.read_stream(i, dataUnion.charVet, CHUNKS_SIZE);
```

```
        displayDataUnionMPU(&dataUnion);
492     }

494     for (long i=BLOCK1_GPS_IA;i<BLOCK1_GPS_FA;i=i+(CHUNKS_SIZE*2))
        {
496       SpiRam.read_stream(i, dataUnion.charVet, CHUNKS_SIZE*2);
          displayDataUnionGPS(&dataUnion);
498     }
}
500

void displayConfigPage(){
502

    char msg[100], vet[PAGE_SIZE];
504

    Serial.println("\n--> Inicio da primeira pagina");
506   Serial.print("Data (GMT): ");
      Serial.print(readByteEEPROM(crashLineGPS-CHUNKS_SIZE*2+3));
508   Serial.print("/");
      Serial.print(readByteEEPROM(crashLineGPS-CHUNKS_SIZE*2+2));
510   Serial.print("/");
      Serial.println(readIntEEPROM(crashLineGPS-CHUNKS_SIZE*2));
512

      Serial.print("Horario (GMT): ");
514   Serial.print(readByteEEPROM(crashLineGPS-CHUNKS_SIZE*2+4));
      Serial.print(":");
516   Serial.print(readByteEEPROM(crashLineGPS-CHUNKS_SIZE*2+5));
      Serial.print(":");
518   Serial.print(readByteEEPROM(crashLineGPS-CHUNKS_SIZE*2+6));
      Serial.print(":");
520   Serial.println(readByteEEPROM(crashLineGPS+CHUNKS_SIZE*2+7));
    sprintf(msg,"Pronta para uso? %c",readIntEEPROM(0)); Serial.println(msg);
522 sprintf(msg,"Passou no self test? %c",readIntEEPROM(1)); Serial.println(msg);
    sprintf(msg,"Escala do acelerometro na calibracao: +/- 2g "); Serial.println(
    msg);
524 sprintf(msg,"Escala do giroscopio na calibracao: +/- 250 graus/s"); Serial.
    println(msg);
    sprintf(msg,"Offset do acelerometro: X=%05d Y=%05d Z=%05d",readIntEEPROM(4),
    readIntEEPROM(6),readIntEEPROM(8)); Serial.println(msg);
526 sprintf(msg,"Offset do giroscopio:   X=%05d Y=%05d Z=%05d",readIntEEPROM(10),
    readIntEEPROM(12),readIntEEPROM(14)); Serial.println(msg);
    sprintf(msg,"Escala do acelerometro na operacao: +/- 2 g"); Serial.println(msg)
    ;
528 sprintf(msg,"Escala do giroscopio na operacao: +/- 250 graus/s"); Serial.
    println(msg);
    sprintf(msg,"Limiar de disparo do acelerometro: X=%d Y=%d Z=%d g",readIntEEPROM
    (18),readIntEEPROM(20),readIntEEPROM(22)); Serial.println(msg);
530 sprintf(msg,"Limiar de disparo do giroscopio: X=%d Y=%d Z=%d graus/s",
    readIntEEPROM(24),readIntEEPROM(26),readIntEEPROM(28)); Serial.println(msg);
    sprintf(msg,"Linha de disparo no MPU: %ld",readLongEEPROM(30)); Serial.println(
    msg);
532 sprintf(msg,"Disparo no acelerometro? X=%c Y=%c Z=%c",readCharEEPROM(33),
    readCharEEPROM(34),readCharEEPROM(35)); Serial.println(msg);
```

```
    sprintf(msg,"Disparo no no giroscopio? X=%c Y=%c Z=%c",readCharEEPROM(36),
    readCharEEPROM(37),readCharEEPROM(38)); Serial.println(msg);
    sprintf(msg,"Linhha de disparo no GPS: %ld ",readLongEEPROM(48)); Serial.
    println(msg);
    Serial.println("--> Fim da primeira pagina");Serial.println();

//    sprintf(msg," "); Serial.print(msg);
//    sprintf(msg," "); Serial.print(msg);
}

 void initizalingConfigPage(){
          SpiRam.write_byte((long) 0, (char) crashState);
          SpiRam.write_byte((long) 1, (char) 'S');
          SpiRam.write_int((long) 4,AX_OFFSET_DEF);
          SpiRam.write_int((long) 6,AY_OFFSET_DEF);
          SpiRam.write_int((long) 8,AZ_OFFSET_DEF);
          SpiRam.write_int((long) 10,GX_OFFSET_DEF);
          SpiRam.write_int((long) 12,GY_OFFSET_DEF);
          SpiRam.write_int((long) 14,GZ_OFFSET_DEF);
          SpiRam.write_int((long) 18,SHOOT_AX_G);
          SpiRam.write_int((long) 20,SHOOT_AY_G);
          SpiRam.write_int((long) 22,SHOOT_AZ_G);
          SpiRam.write_int((long) 24,SHOOT_GX_G);
          SpiRam.write_int((long) 26,SHOOT_GY_G);
          SpiRam.write_int((long) 28,SHOOT_GZ_G);
          SpiRam.write_int((long) 30,crashLineMPU);
          SpiRam.write_int((long) 48,crashLineGPS);
 }
```

II. FUNÇÕES PARA USO DA MEMÓRIA EEPROM

```
 /*
The EEPROM 24LC256 has two memory addresses,
one at address 0x50,  with memory addresses from 0..0xffff or( 0..65535)
and the other half  with an I2C address of 0x54,
with memory address from 0..0xffff or (0..65535).

This device has a 128byte write page, therefor any single
write must be limited to not cross a 128 byte address boundry.
i.e  0..127, 128..255, 256..383, 384..511 ...

To follow these limitations, you must either do:
use Wire.write() to write single bytes
write a function that breaks bigger data block across 128 byte boundarys

For this project the function writePageEEPROM writes blocks of 128 bytes,
which avoids crossing the pages boundaries
*/

#include <Wire.h>
#define EEPROM_ADDRESS1 0x50     //Address of first half 24LC256 eeprom chip
#define EEPROM_ADDRESS2 0x54     //Address of second half 24LC256 eeprom chip

void writePageEEPROM(long  address, char* data) {

int eepromAddress;

      if(address<65536)
        eepromAddress=EEPROM_ADDRESS1;
      else
        eepromAddress=EEPROM_ADDRESS2;

      for (unsigned int i=0; i<4;i++) // From 0 to 3
      {

       Wire.beginTransmission(eepromAddress);
       Wire.write((int)((address+(30*i)) >> 8)); // MSB
       Wire.write((int)((address+(30*i)) & 0xFF)); // LSB

            for (unsigned int j=0; j<30;j++) // From 0 to 29
            {
            Wire.write((byte) data[(30*i)+j]);
            }
       Wire.endTransmission();
       delay(6); // needs 5ms for page write
       }

       Wire.beginTransmission(eepromAddress);
```

```
48        Wire.write((int) ((address+120) >> 8)); // MSB
          Wire.write((int) ((address+120) & 0xFF)); // LSB
50              for (unsigned int j=0; j<8;j++) // From 0 to 7
                {
52              Wire.write((byte) data[120+j]);
                }
54        Wire.endTransmission();
          delay(6); // needs 5ms for page write
56
}
58
// Should NOT read more than 30 bytes at a time
60 void readEEPROM(long address, unsigned char* data, unsigned int num_chars) {
  unsigned char i=0;
62  int eepromAddress;
  DataUnion dataUnion;
64
  if(address<65536)
66    eepromAddress=EEPROM_ADDRESS1;
  else
68    eepromAddress=EEPROM_ADDRESS2;

70         dataUnion.longVet[0]=address;
  Wire.beginTransmission(eepromAddress);
72  Wire.write((int)(address >> 8)); // MSB
  Wire.write((int)(address & 0xFF)); // LSB
74  Wire.endTransmission();
  Wire.requestFrom(eepromAddress,num_chars);
76  while(Wire.available()) data[i++] = Wire.read();
}
78
int readIntEEPROM(long address) {
80  int eepromAddress;

82  if(address<65536)
    eepromAddress=EEPROM_ADDRESS1;
84  else
    eepromAddress=EEPROM_ADDRESS2;
86
    Wire.beginTransmission(eepromAddress);
88  Wire.write((int)(address >> 8)); // MSB
    Wire.write((int)(address & 0xFF)); // LSB
90  Wire.endTransmission();
    Wire.requestFrom(eepromAddress,2);
92  if(Wire.available()<=2)
      //return(Wire.read()<<8|Wire.read());
94    return(word(Wire.read(),Wire.read()));
    else
96    Serial.println("Erro ao ler inteiro da EEPROM");
  }
98
char readCharEEPROM(long address) {
```

```
100   int eepromAddress;

102   if(address<65536)
        eepromAddress=EEPROM_ADDRESS1;
104   else
        eepromAddress=EEPROM_ADDRESS2;
106
        Wire.beginTransmission(eepromAddress);
108     Wire.write((int)(address >> 8)); // MSB
        Wire.write((int)(address & 0xFF)); // LSB
110     Wire.endTransmission();
        Wire.requestFrom(eepromAddress,1);
112     if(Wire.available()==1)
          return(Wire.read());
114     else
          Serial.println("Erro ao ler char da EEPROM");
116   }

118 byte readByteEEPROM(long address) {
      int eepromAddress;
120
      if(address<65536)
122     eepromAddress=EEPROM_ADDRESS1;
      else
124     eepromAddress=EEPROM_ADDRESS2;

126     Wire.beginTransmission(eepromAddress);
        Wire.write((int)(address >> 8)); // MSB
128     Wire.write((int)(address & 0xFF)); // LSB
        Wire.endTransmission();
130     Wire.requestFrom(eepromAddress,1);
        if(Wire.available()==1)
132       return(Wire.read());
        else
134       Serial.println("Erro ao ler char da EEPROM");
      }
136
    long readLongEEPROM(long address) {
138   int eepromAddress,i=0;

140   union DataUnionTemp
      {
142    long longVet;
       char charVet[4];
144   };

146   DataUnionTemp dataUnionTemp;

148   if(address<65536)
        eepromAddress=EEPROM_ADDRESS1;
150   else
        eepromAddress=EEPROM_ADDRESS2;
```

```
152
    Wire.beginTransmission(eepromAddress);
154   Wire.write((int)(address >> 8)); // MSB
    Wire.write((int)(address & 0xFF)); // LSB
156   Wire.endTransmission();
    Wire.requestFrom(eepromAddress,4);
158   if(Wire.available()<=4){
      dataUnionTemp.charVet[i];
160     i++;}
    else
162     Serial.println("Erro ao ler inteiro longo da EEPROM");
    return (dataUnionTemp.longVet);
164 }
```

III. FUNÇÕES PARA USO DO GPS

```
#include <TinyGPS.h>
// Code modified from https://www.youtube.com/watch?v=scOAzTiOes4

bool getGPSmotion(DataUnion *dataUnion){
bool recebido = false;


  while (Serial2.available()) {
     char cIn = Serial2.read();
     recebido = gps1.encode(cIn);
  }


  if (recebido) {

     digitalWrite(LED_VD,HIGH);

     //Latitude e Longitude
     long latitude, longitude;
     unsigned long dalayInfo;
     gps1.get_position(&latitude, &longitude, &dalayInfo);

     if (latitude != TinyGPS::GPS_INVALID_F_ANGLE) {
       // Serial.print("Latitude: ");
      //  Serial.println(float(latitude) / 100000, 6);
     }else
     {latitude=0;}

     if (longitude != TinyGPS::GPS_INVALID_F_ANGLE) {
       // Serial.print("Longitude: ");
       // Serial.println(float(longitude) / 100000, 6);
     }else
     {longitude=0;}

     if (dalayInfo != TinyGPS::GPS_INVALID_AGE) {
        //Serial.print("Idade da Informacao (ms): ");
        //Serial.println(dalayInfo);
     }else
     {dalayInfo=0;}


     //Dia e Hora
     int ano;
     byte mes, dia, hora, minuto, segundo, centesimo;
     gps1.crack_datetime(&ano, &mes, &dia, &hora, &minuto, &segundo, &centesimo, &
    dalayInfo);

```

```
//      Serial.print("Data (GMT): ");
//      Serial.print(dia);
//      Serial.print("/");
//      Serial.print(mes);
//      Serial.print("/");
//      Serial.println(ano);
//
//      Serial.print("Horario (GMT): ");
//      Serial.print(hora);
//      Serial.print(":");
//      Serial.print(minuto);
//      Serial.print(":");
//      Serial.print(segundo);
//      Serial.print(":");
//      Serial.println(centesimo);


      //altitude
      float altitudeGPS;
      altitudeGPS = gps1.f_altitude();

      if ((altitudeGPS != TinyGPS::GPS_INVALID_ALTITUDE) && (altitudeGPS != 1000000)
    ) {
//          Serial.print("Altitude (cm): ");
//          Serial.println(altitudeGPS);
      }else
      {altitudeGPS=0.0;}


      //velocidade
      float velocidade;
      //velocidade = gps1.speed();         //nos
      velocidade = gps1.f_speed_kmph();    //km/h
      //velocidade = gps1.f_speed_mph();   //milha/h
      //velocidade = gps1.f_speed_mps();   //milha/segundo

     // Serial.print("Velocidade (km/h): ");
     // Serial.println(velocidade, 2);  //Conversao de Nos para Km/h



      //sentito (em centesima de graus)
      unsigned long sentido;
      sentido = gps1.course();

      // Serial.print("Sentido (grau): ");
      // Serial.println(float(sentido) / 100, 2);


  dataUnion->intVet[0]=ano;
  dataUnion->bytesVet[2]=mes;
  dataUnion->bytesVet[3]=dia;
```

```
98    dataUnion->bytesVet[4]=hora;
      dataUnion->bytesVet[5]=minuto;
100   dataUnion->bytesVet[6]=segundo;
      dataUnion->bytesVet[7]=centesimo;
102   dataUnion->floatVet[2]= float (latitude)/100000;
      dataUnion->floatVet[3]= float (longitude)/100000;
104   dataUnion->floatVet[4]=altitudeGPS;
      dataUnion->floatVet[5]=velocidade;
106
      return true;
108   }
      else
110   return false;
  }
112
```

IV. FUNÇÕES PARA USO DO MPU6050

```
#include <Wire.h>

//Offsets for MPU6050 calibration
#define AX_OFFSET_DEF -1630
#define AY_OFFSET_DEF 191
#define AZ_OFFSET_DEF 2574
#define GX_OFFSET_DEF 719
#define GY_OFFSET_DEF 170
#define GZ_OFFSET_DEF -24

#define MPUaddress 0x68 // MPU6050 I2C address
#define SS_PIN 53       // Pin CS

void calibrateMPU6050 (){

long int ax_offset=0, ay_offset=0, az_offset=0;
long int gx_offset=0, gy_offset=0, gz_offset=0;
int ax, ay, az;
int gx, gy, gz;
int temp;

int countLimit=10000;

Serial.println("Carregando...");

for(int i=0;i<countLimit;i++)
  {

  Wire.beginTransmission(MPUaddress);
  Wire.write(0x3B);  // starting with register 0x3B (ACCEL_XOUT_H)
  Wire.endTransmission(false);

  //Solicita os dados do sensor
  Wire.requestFrom(MPUaddress,14,true);

  ax=Wire.read()<<8|Wire.read();
  ay=Wire.read()<<8|Wire.read();
  az=Wire.read()<<8|Wire.read();
  temp=Wire.read()<<8|Wire.read();
  gx=Wire.read()<<8|Wire.read();
  gy=Wire.read()<<8|Wire.read();
  gz=Wire.read()<<8|Wire.read();

  if(((long) i/countLimit)==0.25)
  {Serial.println("25%");}

  if(((long) i/countLimit)==0.5)
```

```
  {Serial.println("50%");}

  if(((long) i/countLimit)==0.75)
  {Serial.println("75%");}

// Serial.print("ax= ");Serial.print(ax);
// Serial.print(" ay= ");Serial.print(ay);
// Serial.print(" az= ");Serial.print(az);
// Serial.print(" temp= ");Serial.print(temp);
// Serial.print(" gx= ");Serial.print(gx);
// Serial.print(" gy= ");Serial.print(gy);
// Serial.print(" gz= ");Serial.println(gz);


  ax_offset=ax_offset+ax;
  ay_offset=ay_offset+ay;
  az_offset=az_offset+az;
  gx_offset=gx_offset+gx;
  gy_offset=gy_offset+gy;
  gz_offset=gz_offset+gz;
  }

  ax_offset= -ax_offset/countLimit;
  ay_offset= -ay_offset/countLimit;
  az_offset= -16384-(az_offset/countLimit);
  gx_offset= -gx_offset/countLimit;
  gy_offset= -gy_offset/countLimit;
  gz_offset= -gz_offset/countLimit;

  Serial.print("ax_offset = ");Serial.println(ax_offset);
  Serial.print("ay_offset = ");Serial.println(ay_offset);
  Serial.print("az_offset = ");Serial.println(az_offset);
  Serial.print("gx_offset = ");Serial.println(gx_offset);
  Serial.print("gy_offset = ");Serial.println(gy_offset);
  Serial.print("gz_offset = ");Serial.println(gz_offset);

  delay(300);

}

void getMPU6050motion (DataUnion *dataUnion){


  Wire.beginTransmission(MPUaddress);
  Wire.write(0x3B);  // Starting with register 0x3B (ACCEL_XOUT_H)
  Wire.endTransmission(false);

  //Request data from MPU
  Wire.requestFrom(MPUaddress,14,true);

  //Store accelaration and gyro values on the variables
```

```
99    dataUnion->int16Vet[0]=(Wire.read()<<8|Wire.read());   //0x3B (ACCEL_XOUT_H) & 0
        x3C (ACCEL_XOUT_L)
      dataUnion->int16Vet[1]=(Wire.read()<<8|Wire.read());   //0x3D (ACCEL_YOUT_H) & 0
        x3E (ACCEL_YOUT_L)
101   dataUnion->int16Vet[2]=(Wire.read()<<8|Wire.read());   //0x3F (ACCEL_ZOUT_H) & 0
        x40 (ACCEL_ZOUT_L)
      dataUnion->int16Vet[7]=(Wire.read()<<8|Wire.read());   //0x41 (TEMP_OUT_H) & 0x42
        (TEMP_OUT_L)
103   dataUnion->int16Vet[3]=(Wire.read()<<8|Wire.read());   //0x43 (GYRO_XOUT_H) & 0x44
         (GYRO_XOUT_L)
      dataUnion->int16Vet[4]=(Wire.read()<<8|Wire.read());   //0x45 (GYRO_YOUT_H) & 0x46
         (GYRO_YOUT_L)
105   dataUnion->int16Vet[5]=(Wire.read()<<8|Wire.read());   //0x47 (GYRO_ZOUT_H) & 0x48
         (GYRO_ZOUT_L)

107 }
```